

经济管理精品教材

金融学系列

Quantitative Investment Using Python

量化投资与Python语言

张 翔 著

清华大学出版社

量化投资与Python语言

张　翔　编著

清华大学出版社
北　京

内 容 简 介

Python 是当前金融行业的主流编程语言，金融机构特别是量化投资领域大量使用 Python 进行数据分析以及投资策略测试、实盘交易等。财经类院校基本上没有开设 Python 编程这门课，主要还是 Excel、R 等。

本书主要包括三部分：介绍金融领域内的前沿科技，主要是大数据、云计算、人工智能等；二是 Python 数据分析篇，主要介绍 Python 编程基础，Pandas 数据分析以及网络爬虫；三是量化投资篇，主要包括量化投资常见策略，当前国内量化投资平台简介，平台策略开发案例分析等。

本书适合财经类院校的学生、金融机构的从业人员学习，上手简单，有助于在大数据背景下的各种金融投资技术的应用开发。

图书在版编目(CIP)数据

量化投资与 Python 语言/张翔著. —北京：清华大学出版社，2018
(21 世纪经济管理精品教材·金融学系列)
ISBN 978-7-302-48956-6

Ⅰ. ①量…　Ⅱ. ①张…　Ⅲ. ①投资－软件工具－程序设计－高等学校－教材　Ⅳ. ①F830.59-39

中国版本图书馆 CIP 数据核字(2018)第 294077 号

责任编辑：陆浥晨
封面设计：李绍霞
责任校对：宋玉莲
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市君旺印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：7.75　　**字　　数**：160 千字
版　　次：2018 年 6 月第 1 版　　**印　　次**：2018 年 6 月第1 次印刷
定　　价：35.00 元

产品编号：075559-01

写在前面的话

我本科和研究生都是学数学的，和朋友见面，很多时候知道你是数学专业的，都会觉得你好聪明，数学不是一般人能学习的。说实话，一直以来，我都认为数学和其他学科没什么两样，只是每个人口味不同而已，一些人喜欢物理，一些人喜欢土木工程；就像在农村学做个砖匠、木匠或者瓦匠一样，只是一门技艺。大部人学习一门专业，其实更多的是一种方法论的学习，专业修养的积累；同时，做着文化传承的工作。当然，肯定也会有一些有兴趣或天赋的人一生致力于数学某一个领域的研究，比如陈景润。

那么对于文科背景的人在当今这样的时代，怎样快速掌握金融领域内的科技以及量化金融工作方法呢？我认为方法论就很重要。比如我学习的是数学，大学时期就开始做家教，办培训班，对于数学我会看重三点：一是数学研究某一事物一般遵循三个步骤，首先把要研究东西的放在一起，也就是要定义一些东西；接着就要研究基于这个定义的这些东西有什么性质，如果是代数方面的就是有哪些运算规则；最后就是这些性质对于我们生活有什么用，怎么解决现实的问题。二是搞数学就是搞关系，我们搞清楚了这个东西的性质，那个东西的性质也搞清楚了，它们之间有什么关系呢，某一个定理或者结论总是要解释事物间的关系。三是我们总是在受约束的条件下解决一个最优化的问题，不管是搞学术还是工作、生活都面临一个约束条件，我们要认识到这个约束条件，然后去讨论可行性，找出最优化的方案。

写这么多其实就是想说，我接下来介绍的金融科技都是基于这样的理念去阐述的，希望大家掌握一个模型，快速掌握这些信息，至少让我们在这个行业有一幅“地图”，知道应该在哪个方面去最大化自己的能力和禀赋，在工作和生活中更加游刃有余。



当我们做一件事情或说一句话的时候都是有假设条件的，我们没有口头提出是因为如果我们每次说话都要像做数学题一样，先把假设条件说一下，再沟通也确实太闷或者无味了。

我想必须是要有前言来说明一下写这本书的动机，而这个动机的背景、假设以及定位也要在这里做个具体的介绍。

还得从2010年股指期货开通说起，股指期货的推出使得量化投资开始兴起，CTP（综合交易开发平台）的推出使得很多从事软件行业或者理工科背景的“牛人”纷纷转行进入股指期货交易这个领域，当然那几年国内很多外资背景的电子行业的公司撤出中国也导致很多计算机行业从业人员转行过来，这和《我的宽容人生》里的主人公所处背景差不多：美国“冷战”结束，在军工方面的经费投入急剧减少，使得很多火箭专家纷纷投入华尔街。新兴事物的发展总是由各种机缘促成的，这几年大数据、云计算和区块链等科技在大众创业、万众创新的潮流中精彩纷呈，从华尔街回来的“80后”创建自己的对冲基金，量化对冲基金逐渐成为国内的主流。

总之，科技在改变生活，而离得最近、最值得挑战的就是金融市场，我们现在从事金融行业，研究量化投资和量化技术。

这本书面向的读者是财经院校的大学生或者研究生，所以我不会讲很多知识点，不会讲得很复杂，尽可能讲得清楚。我认为本书更像是一幅地图，能让读者迅速了解这个行业，能掌握一个模型，快速上手去实战或参与项目，然后不断深入。

本书可以作为培训教材，其定位也是希望让非财经院校背景的学生毕业后能迅速了解当前金融市场的生态，能在金融机构迅速上手做项目或成为一名基金经理助理；能让金融从业人员或者投资者更好地使用量化投资工具进行数据分析，从而更好地进行投资。

同时，本书不会花时间去介绍投资组合、风险管理等理论，更多强调的是使用Python，而且假设都是使用Windows系统，我们不会告诉读者要在Linux或者Mac上怎么做，我们仅仅提供一个系统的解决方案，让读者快速



当我们做一件事情或说一句话的时候都是有假设条件的，我们没有口头提出是因为如果我们每次说话都要像做数学题一样，先把假设条件说一下，再沟通也确实太闷或者无味了。

我想必须是要有前言来说明一下写这本书的动机，而这个动机的背景、假设以及定位也要在这里做个具体的介绍。

还得从2010年股指期货开通说起，股指期货的推出使得量化投资开始兴起，CTP(综合交易开发平台)的推出使得很多从事软件行业或者理工科背景的“牛人”纷纷转行进入股指期货交易这个领域，当然那几年国内很多外资背景的电子行业的公司撤出中国也导致很多计算机行业从业人员转行过来，这和《我的宽容人生》里的主人公所处背景差不多：美国“冷战”结束，在军工方面的经费投入急剧减少，使得很多火箭专家纷纷投入华尔街。新兴事物的发展总是由各种机缘促成的，这几年大数据、云计算和区块链等科技在大众创业、万众创新的潮流中精彩纷呈，从华尔街回来的“80后”创建自己的对冲基金，量化对冲基金逐渐成为国内的主流。

总之，科技在改变生活，而离得最近、最值得挑战的就是金融市场，我们现在从事金融行业，研究量化投资和量化技术。

这本书面向的读者是财经院校的大学生或者研究生，所以我不会讲很多知识点，不会讲得很复杂，尽可能讲得清楚。我认为本书更像是一幅地图，能让读者迅速了解这个行业，能掌握一个模型，快速上手去实战或参与项目，然后不断深入。

本书可以作为培训教材，其定位也是希望让非财经院校背景的学生毕业后能迅速了解当前金融市场的生态，能在金融机构迅速上手做项目或成为一名基金经理助理；能让金融从业人员或者投资者更好地使用量化投资工具进行数据分析，从而更好地进行投资。

同时，本书不会花时间去介绍投资组合、风险管理等理论，更多强调的是使用Python，而且假设都是使用Windows系统，我们不会告诉读者要在Linux或者Mac上怎么做，我们仅仅提供一个系统的解决方案，让读者快速

学习并掌握方法，通过掌握现代金融理论展示自己的能力。

时势造英雄。希望读者乘着多层次金融市场建设和创新创业的大风，临风起舞，驭风而行！

编　者


目
录

第1章

金融科技

1.1 谈谈方法论

大家都知道现在是大数据时代,到处都是数据,结果就是数据到处都是,但有用的信息却越来越少了。怎样从大量繁杂的数据中得到有效的数据或者怎么去利用这些数据,需要方法论的指导。

比如,现在很多场合,看微信朋友圈成了最好的社交应对策略。寂寞无聊的时候看看朋友圈,走在路上看看朋友圈,人多的时候不想说话了,看看朋友圈。朋友圈的信息真的是囊括万千,应有尽有。在信息碎片化的时代,我们很容易迷失,以为自己了解了很多,其实真要让自己做个整理才发现,好像什么都不清晰。其实,我们要的不是所有权,而是使用权;我们要的不是大量的信息,而是利用这些信息的能力。

写这本书的第一个目的就是要解决这样的问题,我们能利用现有的信息,更好地展示我们的价值。

记得当年比较火的一部电视剧叫《射雕英雄传》,闲暇时间又看了一遍,郭靖这个形象当然很是鲜明,但更引起我关注的还是武侠小说里面的全真派的心法和各门派武功的关系了,郭靖在学习了全真派的心法后武学突飞猛进,后来学习各种武功都能融会贯通,这就像我们要学习国学打好基础,再学习其他科目;或者掌握方法再学习其他技能一样。总之要处理好内容与形式的关系,做金融行业更是如此。

现在社会上开始倡导读书,图书出版市场也算是火起来了,图书价格也在加速上涨。我认为图书可以分为以下四类。

(1) 有营养,有包装的书。

(2) 没有营养,有包装的书。

(3) 有营养,没有包装的书。

(4) 没有营养,没有包装的书。

比如中国四大名著就是有营养的书,而有一些卖“鸡汤”的书翻了一遍后都不知道究竟要表达什么,就像吃方便面一样,确实也解决了一些问题,但总是觉得没营养。

我希望读者看完这本书之后,能火眼金睛地选择你要读的书,并快速地吸收知识,转化成你需要的实用的信息并应用于实践。

现在这个时代,各种专业术语以及网络术语每天都在更新,稍不注意可能都不知道对方在做什么。

前几年,大家都在谈 P2P,说互联网金融,从 2016 年开始,大家又纷纷提及的词是 fintech,也就是金融科技。

今年这个词更加泛滥了,好像不提金融科技,就不是在做创新,不是在做金融行业。

可问题就来了，假如隔壁的二娃问你，科技金融和金融科技有什么区别呢？可能你会觉得，哦，好像没想过，然后你就支支吾吾，如果你还在金融圈混，想想你的形象是不是一下就弱了很多呢？

举个例子，假设你是一个做大数据的技术“牛人”，有一次高中同学聚会，大家在一起聚餐，然后隔壁的老王夹了口菜说好吃啊，又夹了口菜说这个也很好吃啊；然后你突然放下筷子半开玩笑半认真地说，其实我认为这个世界上最有营养的菜有两种，这个时候可能你身边的翠花、油菜花还有小明都没回应说来听听呢，然后你说第一种就是最难闻的，另外一种就是最难吃的。这个时候可能他们就突然会发现，哇，原来你还是很懂养生之道嘛，好像美食专家一样，而事实上，可能你就是一个宅男，无非就是运用了二分法，而这种方法在数学以及很多学科上应用得太多了。

1.2 金融、科技和互联网

现在各种术语满天飞，有时候我们可能都不知道到底是什么意思，这就为我们塑造价值提供了机会，我们得有策略地告诉对方，我们对这些知识和前沿方向掌握得更好、更专业。

互联网金融、科技金融和金融科技其实不就是金融、科技和互联网的名词组合嘛，那么我们还得顾名思义，至少这是思考的很重要的第一步。

互联网金融和科技金融强调的是金融，互联网和科技只是一个定语。很多学科都需要这么简单的一个道理的建立，比如司法考试(2018 年就是法律职业资格考试)在刑法这方面，各个罪名的解释就是一个很重要的技术。

比如，有个小白问你，互联网金融是什么呢？我们怎么回答呢，你可以直接说就是互联网时代的金融嘛，当然这样确实也是太闷了，我们适当包装一下就好了，我们可以这样讲：互联网金融，顾名思义就是互联网情景的新兴金融业务模式。甚至，还可以加一句“在美国这样的发达国家没有互联网金融”这种说法的。这样的回答，至少让对方觉得你举重若轻，很专家的样子。

同时，最勾魂的是，你要互动，要让对方参与，让对方真正觉得你专业的还是你提问的方式。比如，你可以向对方抛出一个话题：你知道互联网金融的核心是什么？

或者，你也可以这么问：你知道是什么让互联网和金融搞在一起的吗？

对方可能会问：你说的是 P2P、大数据、互联网、民间理财这些吗？对方可能会把他接收到的信息或者概念很凌乱地抛给你。

这个时候，你要显得很专业地说：其实就是两个字，支付。

对方一听，忽然觉得，很有道理啊。

下面这些就是你发挥的依据了。

无支付不金融。

金融的最基本功能是存、贷、汇，即货币资金的聚集功能、运用功能、支付功能。随着现代金融的发展，三个功能中的聚集功能、运用功能都得到了快速发展，演化出了不同的银行、证券、期货、保险、信托、基金等金融机构承担并实现其功能，只有支付功能没有得到

充分发展并制约了金融功能的发挥、实现、深化、创新以及对居民个人、组织交往和实体经济提供服务。

在市场经济时代，经济和社会活动，很多都需要完成交易，促成交易完成的"最后一公里"是支付。虽然传统金融组织、市场、产品创新都比较快，而支付组织、支付工具、支付形式和支付功能的实现，与互联网时代的支付需求拉开了差距，从而促成了第三方支付的发展。

在第三方支付产生以前，支付清算体系是客户与商业银行建立联系，商业银行与中央银行建立联系，中央银行是所有商业银行支付清算的对手方，能够通过轧差进行清算。在原有支付清算模式下，由于客户不能与中央银行之间直接建立联系，客户必须分别与每一家商业银行建立联系，支付清算的效率较低。

第三方支付诞生以后，客户与第三方支付公司建立联系，第三方支付公司代替客户与商业银行建立联系。这时第三方支付公司成为客户与商业银行支付清算的对手方，第三方支付公司通过在不同银行开立的中间账户对大量交易资金实现轧差，少量的跨行支付则通过中央银行的支付清算系统来完成。

那么问题又来了，传统支付和第三方支付有什么区别呢(如图 1-1 所示)？

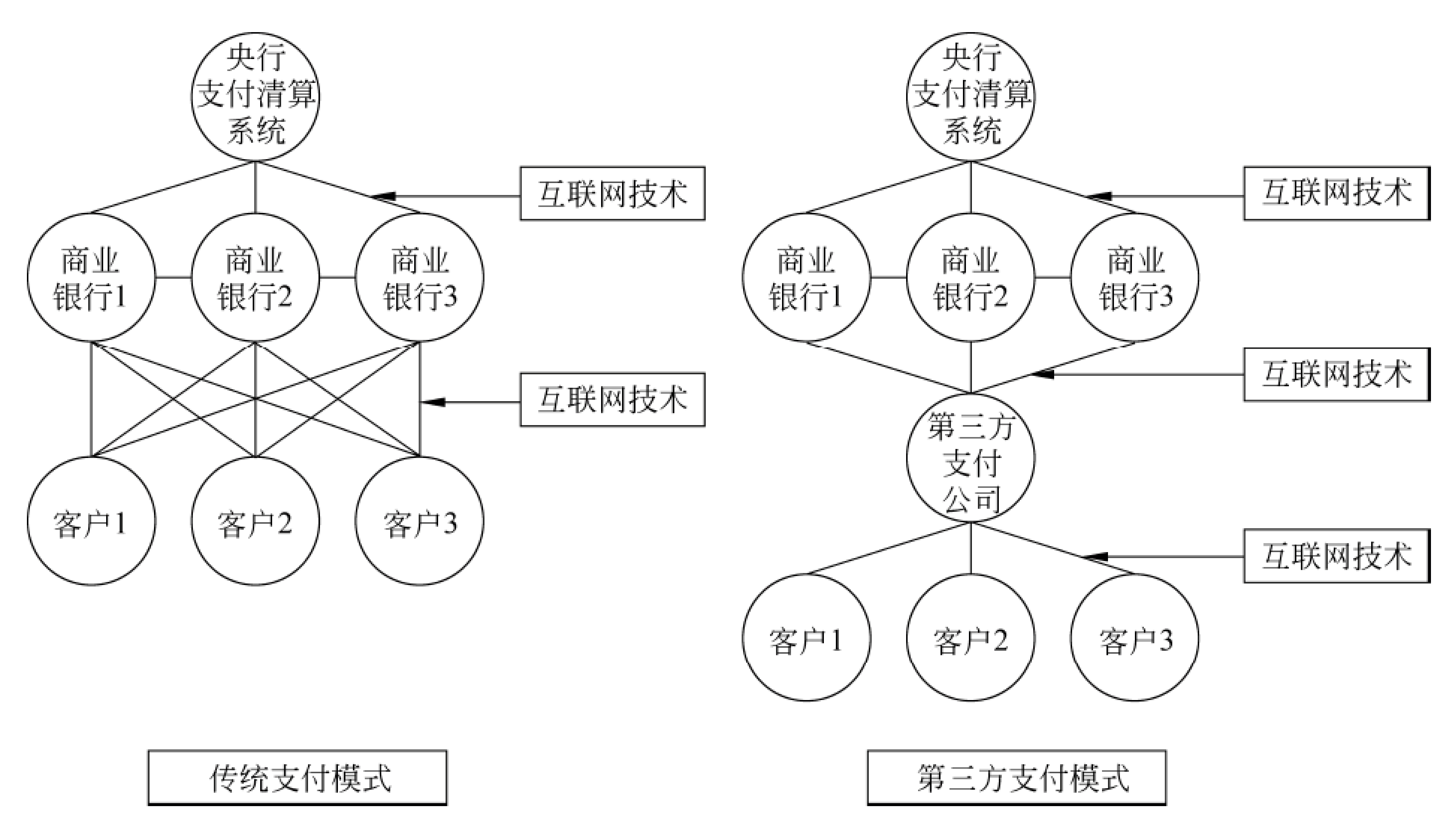


图 1-1 传统支付与第三方支付的区别

第三方支付通过采用二次结算的方式，实现了大量小额交易在第三方支付公司的轧差后清算，在一定程度上承担了类似中央银行的支付清算功能，同时还能起到信用担保的作用。

在移动支付产生以前，客户与第三方支付公司建立连接主要通过计算机端实现，移动支付诞生以后，客户与第三方支付公司的联系逐渐向手机端转移。

第三方支付现在体现的形式主要就是移动支付。

移动支付主要指通过移动通信设备、利用无线通信技术来转移货币价值以清偿债权债务关系。近年来我国移动支付发展迅速，移动支付的形式更加多样化，出现了短信支

付、NFC 近场支付、语音支付、二维码扫描支付、手机银行支付、刷脸支付等多种支付方式。

我国的移动支付模式，如果是由银行推出，则需要开通手机银行，同时为了配合近场支付，可能还需要手机具有 NFC 功能。如果是三大运营商推出的移动支付，一般是通过在 SIM 卡植入芯片来完成支付（如手机贴膜卡、翼支付的 RFID-UIM 卡等）。如果是纯粹的第三方支付公司推出，可以不用开通手机银行，就可直接进行支付，如支付宝的“碰碰刷”、微信支付等，其特点是方便快捷，最大限度地满足客户对速度的要求。但第三方支付公司推出的移动支付，安全性不及手机银行，多数情况由保险公司来进行承保。

这样一来，我们对互联网金融就有了一个很贴切的理解。任何时候我们都可以很好地向他人去解释什么是互联网金融了。而且还可以进一步谈谈第三方支付、传统支付和移动支付。

这里我们重点讲了互联网金融，而且强调了互联网金融的关键词：支付。

科技和金融的组合也可以同样处理。

我们以蚂蚁金服为例，它之所以将自己定义为一个金融科技公司，主要是因为它是利用萌生于金融本身的大数据、区块链来对金融本身进行改造，从而产生一种已经发生本质改变的事物，这种事物与传统金融有实质性的区别。

几年以前，科技和金融可能只是一种简单的组合关系，两者之间并未发生深度融合。而随着移动互联网的发展，金融和科技发生了深度的融合。

借助科技，日常生活中越来越多的场景正在“金融化”，而借助金融与互联网，科技企业的发展也更为迅速；随着越来越多的金融服务场景通过科技更便捷地呈现出来，更多的互联网金融企业也越往“金融科技”的战略方向发展。

1.3 金融科技概念

1. 金融科技的核心

金融科技（fintech）已经成为金融行业最时髦的话题。“fintech”到底是什么，又为何在中国火了？

同样的，假设二娃问你，金融科技到底是什么，怎么回答好呢？你可以回答，金融科技就是科技在金融中的应用；或者你回答：用科技的手段改造金融行业。

其实你会发现，我更强调的是一种回应的技术，毕竟沟通很重要，包装也很重要，这样回答如果你觉得还不过瘾，你还可以引用维基百科的定义：它是由一群通过科技，让金融服务更高效的企业构成的一个经济产业。fintech 公司通常是那些尝试绕过现存金融体系，而直接触达用户的初创企业，它们挑战着那些较少依赖于软件的传统机构。

金融科技的核心是什么呢？或者要解决的问题最明显的特征是什么呢？

答案是信息和信用成本。

金融作为一个现代庞大的支柱产业，其实是指金融服务业，也就是帮助资金融通的服务行业。这个行业从业人员所做的事情和所创造的价值，主要就是让资金从相对闲置、低效率的人或机构手中，流动到可以更高效率使用、产生更大价值的地方。你每个月领工

资，有一部分暂时花不完，借给急需用钱投资或消费的人，就是创造了价值。

庞大的金融服务业，所做的事情其实都是在做“信息”处理。

套用一句广告词：我们不生产信息，我们只是信息的搬运工。

金融业整个工作的对象就是信息，它的产品就是信息。

如今，互联网时代成长起来的我们，每天都习惯了使用支付宝、微信支付和各种电子货币、代币，这些都可以理解成“信息”。

当然，最早的资金可不是信息形态，而是石头贝壳、真金白银等实物形态。

金融业的萌发，可以说肇始于对真金白银等实物资金形态的“信息化”。最早帮助存储金银的金匠铺，开始用纸张等材质记录的票据代替金银给付物主，之后用来流通的时候，其实就是一个“信息化”的过程。

当然，当纸币这种现金形态已经大范围转换为比特的电子、数字货币形态后，大家更容易理解这个信息化过程。这个对资金形态的信息化过程，极大地降低了人们日常的生活和商业交易成本，扩大了市场交易的规模，深化了人们劳动角色的分工，从而也进一步促进了科技的进步、人类生产力的发展和财富的积累。

这是金融服务业对推动社会进步的巨大贡献。但是，在这个过程中，金融业的发展，需要持续解决本身带来的两大问题：信息成本和信用成本。

说到信息，就要说到信息的收集、传递/交互、存储和处理等功能。信息的收集、传递/交互、存储和处理，都需要消耗成本，而且这个成本往往还是巨大的。伴随着资金的“信息化”过程，又带来一个新的问题：信息的真实性。

把我存的黄金用一张纸作为证明代替，你会不会把黄金拿去挥霍掉了，等我要用的时候，两手空空？

这个就是“信用”问题。信息化后，要取得信任、达成交易，还需要付出“信用成本”。你看到的那些豪华气派的银行高楼大厦，就是一种很直观的信用成本。为什么那么多线下理财公司都喜欢将办公室租在市中心高级写字楼的高层？因为这些都是“信用成本”。

金融业的发展过程，可以简单地理解为，就是一个不断降低金融信息成本和信用成本的过程。降低金融信息和信用成本，靠的就是人类的持续创新。

由此可见，金融科技本身不是什么新事物。人类历史上已经出现了无数的金融科技：当把纸张、印刷技术用于纸钞印刷时，这是金融科技；当你不需要背着很重的黄金白银行走江湖或出差，通过银行汇款就可以到处做生意时，这是金融科技；当你不用出门、只要一部手机就可以炒黄金、买美元、躺着理财赚钱时，这也是金融科技。

金融科技之所以现在大热，是因为最近几十年最热的科技是信息科技，而金融服务本质上是信息服务，金融与科技天生具有巨大的相互吸引力。当信息技术和金融服务，这两个最近几十年来发展最快的行业发展到特定时段时，拐点就出现了。

或许有一定的企业换马甲避险、资本推高估值和媒体借势炒作的成分，但是，金融科技的确迎来了发展的黄金时代。

明白了上面的内在逻辑，你就很容易理解那些名目繁多的 fintech 概念是怎么回事了：移动互联网、物联网降低了金融信息收集和传递的成本；大数据和云计算提升了信息处理的效率；智能投顾节省了你去学习、分析和判断大量金融信息的时间和知识成本；区

块链技术应用可通过取消传统中心化的信任中介，大幅降低信用成本；人脸识别等生物识别技术，可通过刷脸验证身份，实现远程开户和操作，降低了庞大的信用成本、时间成本、金融机构大量营业机构和人员的运营成本。

循着这个降低金融信息和信用成本，提高金融服务效率的内在逻辑，你甚至能够预测到，未来有哪些信息技术会应用到金融领域，成为新的"金融科技"——比如，VR 和 AR 技术成熟后，将来通过虚拟现实和可视化操作，可降低信息的交互成本，提升交互体验，并且可以取代大量金融机构的前端服务人员的人工成本。

总而言之，金融服务业就是在自身利益的驱动下，通过最新的信息技术，搭建起金融信息的流动渠道，不断降低资金流动的成本、扩大资金触达的范围。所以，互联网金融也好，金融科技也好，无非是利用最新的互联网等信息技术，做本质上跟传统金融机构同样的事情。

而在这个过程中，对于我们普通消费者来说，由于金融服务带来的成本，在我们的日常支出中占据了很大一块份额，金融科技创新降低了我们需要支出的成本，也就意味着，我们的收入和财富可以用在更多的消费上，即意味着我们的财富增长了。

2. 科技金融的核心

科技金融的百度定义为科技金融属于产业金融的范畴，主要是指科技产业与金融产业的融合。科技金融的主题是金融，其实就是科技应用在金融中，金融本质也是一种服务，可以说是当前很重要的功能，最重要的体现就是效率和品质。

以往，银行是"二八"法则的最忠实拥趸，他们坚信 20%的客户能创造 80%的利润，因此经营重心大多放在中高端客户吸储、理财与向企业大客户放贷上，希望借此来降低经营成本，提升经营收益。银行对于中小微企业、个人贷款以及普通客户的业务往往不够重视。因为对于银行来说，这些小客户就是"长尾"，投入过多的人手和精力去处理"长尾"客户的业务，只会增加银行的成本。最近几年，这一部分客户被互联网金融所吸纳，余额宝就是一个很好的例子。这是第一款意识到了"长尾"客户价值的产品，单个用户的购买量虽小，但庞大基数成就了余额宝的奇迹，让与之合作的天弘基金在不到一年的时间里"山鸡变凤凰"。金融业"长尾"市场表现为高度碎片化，用户数量庞大且个体资金量小，但通过技术手段进行整合和汇集，无疑将产生巨大价值。当这部分"长尾"用户的客户端使用黏性增加时，银行自然可以通过网络来进行推广和营销。

我们可以从以下三方面来看。

(1) 供应链金融和消费金融。比如大家熟悉的马云做的事其实就是解决了金融服务对象的"长尾"现象，就是说在中国，很多科技公司做的事情就是做了一个中间的技术平台，上端去连接这种小 B 或者是 C 端，它通过一些场景，比如一些核心企业、一些渠道，把这些场景全部连接起来，然后通过自建的技术系统把这些分散的资产集合起来，迅速地转化成为标准化的金融机构能够认可的资产，当资产聚集到一定量之后，就可以对接资金端及金融机构。

(2) 科技金融对社会的价值，表现在金融消费服务升级上。其实在金融消费里面无非是提高你的需求匹配效率，就是高精度的匹配，可以节省你的时间，比如不需要到银行去排队了。

（3）更好的、更高品质的金融服务，现在都流行说“把时间花在美好的事物上”，比如，智能投顾就是一个专业的、高品质的金融服务。金融服务的本质，一个是提高效率，另一个是提高品质，这两个都是技术所擅长的。

1.4 金融科技与互联网金融的关系

现在有两种极端的误区：一是认为“金融科技”基本等同于“互联网金融”，原因前面已经说明，两者侧重点有所不同；二是“互联网金融”与“金融科技”泾渭分明。

互联网金融与金融科技的区别到底在哪儿？我们应该轻松地讲解两者之间的关系。

互联网金融是将传统金融机构的产品、购买方式从线下搬到了线上，从本质上来看，在互联网金融条件下，金融机构优化了用户的产品体验、购买渠道、消费方式，对于金融本身来讲并未产生实质性的变化。

互联网金融对行业的影响有限，金融科技却不同。当下互联网金融之所以会出现诸如此类的问题，其中一个很大原因就是互联网对于传统金融行业的改造并不彻底，甚至可以说并没有对金融行业产生实质性的影响。这就导致了很多传统金融已经出现的，如信息不透明、收益难保障、监管难实现等问题，在互联网金融时代依然会出现。

写到这里，读者可能知道怎么给对方去讲解了，我们可以说互联网金融注重的是形式的体验，金融科技强调的是内容的变革。其实，知识本无所谓的多少，也本无所谓的专家，需要的是我们更好地在沟通中去切分罢了。毕竟，不是我们每个人都要去做学术，都要去做更深入的调研和发表论文。

当然，我们还可以换一种说法。

互联网金融是一种商业模式，金融科技是一种金融产品。

如果将互联网金融看作是一种全新的商业模式的话，那么金融科技则是一种全新的金融产品。

比如，我们不难发现，很多互联网金融企业都是将传统金融行业的一些业务转移到了线上，传统模式中需要到金融机构柜台办理的相关业务，现在动动手指即可实现。

金融科技则不一样，它是基于金融本身萌发出的一种全新的产品。这种产品主要是利用大数据、区块链等互联网创新技术进行风险管理与控制，降低成本，提升效率，并从金融本身作出改变，从而再借助互联网的力量给用户全新的体验。

读者理解了这样的方法论，应该知道怎么去更快地学习新的知识以及切分知识了。

那么还可以说互联网金融是发展阶段，金融科技是最终目的。

互联网金融只是将传统金融机构获取用户的方式转移到了线上，它主要借助的是当下互联网的流量红利，以破解或遏制当下传统金融机构用户不断减少的问题。金融机构借助互联网获得了线下流失的部分流量，并通过将传统产品转移到线上销售，让更多的用户能够通过手机就能够买到自己所需的产品。

互联网金融除了能够让用户获得轻松便捷的产品体验之外，还能让传统金融机构的门槛最大限度的降低，传统金融机构当中需要严格审核才能参与其中的投资项目，现在只要通过互联网便能够轻松参与。而互联网的用户又趋向于年轻化，这些年轻用户通常是

被传统金融机构淘汰掉的用户，通过互联网金融无疑能够将这些用户收入囊中，并最终获得更多用户。再加上互联网金融的体验较好，则更容易获得这些用户的青睐。以苏宁众筹、聚米众筹、京东众筹为代表的众筹行业飞快的发展速度同样说明了这一点。

无论是体验还是用户，互联网金融作出的改变仅仅是从用户端进行的，而即便是互联网金融对传统的金融产品进行了简单的拆分和组合，但是这些拆分和组合稍显机械和简单，而且并没有改变金融产品的本质。我们仅仅能够将互联网金融看作是互联网条件下的金融，金融的本质并没有发生太大改变，它依然面临着诸多问题和弊端。互联网对于金融行业的影响依然停留在表面上，如果我们将互联网金融看作是一种金融类型的话，那么，它充其量只能算作是金融发展的一个阶段，并不能被看作是金融发展的终极阶段。

总结：

大家会发现我们介绍了互联网金融、科技金融以及金融科技，不是着眼于研究的深入，而是从沟通、实用的角度去看待这些信息或者知识。毕竟，对于大部分人来讲，面对的还是沟通，怎么更好地和对方沟通以及传播这些知识。

互联网金融我们强调了支付，通过支付让互联网更好地服务于金融，让金融更好地发挥它的服务功能。

科技金融的关键词是：效率和品质。让科技服务于金融，让金融服务在效率和品质方面产生更好的服务。

金融科技的关键词是：信息和信用成本。科技应用在金融中，创新了金融提供服务的形式，解决了金融服务中的信息成本和信用成本方面的问题。

金融科技和科技金融可以看作是内容和形式的关系，金融是内容，科技是形式。

第 2 章 大　数　据

2.1　大数据的定义

如果有人问你到底什么是大数据，可能你也不一定很容易解释清楚吧，我的意思是至少让对方觉得你很专业。至少我们不能说就是大的数据，或者说我用 Excel 操作了几十万的数据，也觉得是做的大数据。

这里我们仍然可以从分类的思想进行解释。

我们面对这个纷繁复杂，充满竞争的世界，需要一种收智商税的能力，那就是分类的能力。

想想中国古代，周易是一个很先进的分类方法，把这个客观世界进行了 64 种分类。《易传·系辞上传》："易有太极，是生两仪，两仪生四象，四象生八卦。"我们可以看作是古人把信息做了一个分类，初步分成了 8 种。

关于一叶障目和一叶知秋。"一叶障目"相当于是用小样本分析近似推理，而真理可能存在于全样本的海量数据之中，借助大数据则可完全克服。"一叶知秋"其实就是反过来说的，小样本可以预测大数据。

接下来谈大数据，为什么我们现在才谈大数据呢？这要从两方面来看，一方面是计算机技术的发展，特别是存储技术和计算技术的发展，以前不能存储的数据现在可以存储了，现在的科学计算技术更多更深入了，运行、计算速度越来越快；另一方面是信息爆炸式增长。而这两方面就像 DNA 一样是螺旋式增长的，是相互促进、相互影响的。

这么来看大数据就很清晰了，为什么提大数据，大数据怎么定义就很清楚了。

维基百科中只有短短的一句话："巨量资料（big data），或称大数据，指的是所涉及的资料量规模巨大到无法通过目前主流软件工具，在合理时间内达到撷取、管理、处理并整理成为帮助企业经营决策更积极目的的资讯。"

维基百科的定义是从大数据的特征入手的。

我们还可以自己定义，比如按照大数据的功能进行定义：大数据是在多样的或者大量数据中，迅速获取信息的能力。

2.2　大数据和统计的关系

其实不管是大数据还是数学、物理或者统计都是对大自然和社会的一种探索而做的一种分类。事实上，没有任何学科是独立的，很多工作和研究都是跨专业的。大数据，也是一个跨专业的领域，主要包括了计算机科学、统计学、数学、语言学等。我们要把大数据和统计、计算机之间的关系做个区别也是一种权宜之计，大数据不是统计，大数据与计算

机有关。

关于大数据和统计的关系，我们可以简单地做这样的一个比较，如表 2-1 所示。

表 2-1 大数据和统计的关系

大 数 据	统 计
问题驱动：分布理论—概率保证—总体推断	数据驱动：实际分布—总体特征—概率判断
因果分析	相关分析
样本抽选	总体描述
数据烟囱	数据平台
问卷调查	接触点
数据精准	数据黑洞
以小见大	以大见小
归纳推理	演绎推理

2.3 大数据的特征

业界通常用 4 个 V(即 volume、variety、value、velocity)来概括大数据的特征。这四个特征其实主要也是从大数据本身的特点来进行分类或者解释的。

一是数据体量巨大(volume)。截至目前，人类生产的所有印刷材料的数据量都是 200PB(1PB=210TB)，而历史上全人类说过的所有的话的数据量大约是 5EB(1EB=210PB)。当前，典型个人计算机硬盘的容量为 TB 量级，而一些大企业的数据量已经接近 EB 量级。

二是数据类型繁多(variety)。这种类型的多样性也让数据被分为结构化数据和非结构化数据。相对于以往便于存储的以文本为主的结构化数据，非结构化数据越来越多，包括网络日志、音频、视频、图片、地理位置信息等，这些多类型的数据对数据的处理能力提出了更高的要求。

三是价值密度低(value)。价值密度的高低与数据总量的大小成反比。以视频为例，一部 1 小时的视频，在连续不间断的监控中，有用数据可能仅有一两秒。如何通过强大的机器算法更迅速地完成数据的价值“提纯”成为目前大数据背景下亟待解决的难题。

四是处理速度快(velocity)。这是大数据区分于传统数据挖掘的最显著特征。根据 IDC 的“数字宇宙”的报告，预计到 2020 年，全球数据使用量将达到 35.2ZB。在如此海量的数据面前，处理数据的效率就是企业的生命。

除了上述主流的定义外，我们还可以用 3S 或者 3I 描述大数据的特征。

3S 指的是：大小(size)、速度(speed)和结构(structure)。

3I 指的是：

(1) 定义不明确(ill-definite)。多个主流的大数据定义都强调了数据规模需要超过传统方法处理数据的规模，而随着技术的进步，数据分析的效率不断提高，符合大数据定义的数据规模也会相应不断变大，因而并没有一个明确的标准。

(2) 令人生畏(intimidating)。从管理大数据到使用正确的工具获取它的价值，利用

大数据的过程中充满了各种挑战。

(3) 即时(immediate)。数据的价值会随着时间快速衰减,因此为了保证大数据的可控性,需要缩短数据搜集到获得数据洞察之间的时间,使得大数据成为真正的即时大数据,这意味着能尽快地分析数据对获得竞争优势至关重要。

2.4 大数据技术的架构

在理解了大数据的特征后,就很容易简化地来理解大数据技术的核心,即大数据的总体架构包括三层:数据存储、数据处理和数据分析。类型复杂和海量由数据存储层解决,快速和时效性要求由数据处理层解决,价值由数据分析层解决。

数据先要通过存储层存储下来,然后根据数据需求和目标来建立相应的数据模型和数据分析指标体系,以对数据进行分析产生价值,而中间的时效性又通过中间数据处理层提供的强大的并行计算和分布式计算能力来完成。三层相互配合,让大数据最终产生价值。

1. 数据存储层

数据有很多分法,有结构化、半结构化和非结构化;也有元数据、主数据和业务数据;还可以分为GIS、视频、文件、语音、业务交易类各种数据。传统的结构化数据库已经无法满足数据多样性的存储要求,因此在RDBMS基础上增加了两种类型:一种是HDFS可以直接应用于非结构化文件存储;一种是NoSQL类数据库,可以应用于结构化和半结构化数据存储。

从存储层的搭建来说,关系型数据库,NoSQL数据库和HDFS分布式文件系统三种存储方式都需要。业务应用根据实际的情况选择不同的存储模式,但是为了业务的存储和读取方便性,我们可以对存储层进一步地封装,形成一个统一的共享存储服务层,简化这种操作。从用户来讲并不关心底层存储细节,只关心数据的存储和读取的方便性,通过共享数据存储层可以实现在存储上的应用和存储基础设置的彻底解耦。

2. 数据处理层

数据处理层核心解决问题在于数据存储出现分布式后带来的数据处理上的复杂度,海量存储后带来了数据处理上的时效性要求,这些都是数据处理层要解决的问题。

在传统的云相关技术架构上,可以将Hive,Pig和Hadoop-MapReduce框架相关的技术内容全部划入到数据处理层的能力上。原来思考的是将Hive划入到数据分析层能力上不合适,因为Hive重点还是在真正处理下的复杂查询的拆分及查询结果的重新聚合上,而MapReduce本身又实现真正的分布式处理能力。

MapReduce只是实现了一个分布式计算的框架和逻辑,而真正的分析需求的拆分,分析结果的汇总和合并还是需要Hive层的能力整合。最终的目的很简单,即支持分布式架构下的时效性要求。

3. 数据分析层

最后回到数据分析层,分析层重点是真正挖掘大数据的价值所在,而价值的挖掘核心又在于数据分析和挖掘。那么数据分析层核心仍然在于传统的BI分析的内容,其包括数

据的维度分析、数据的切片、数据的上钻和下钻、Cube 等。

谈了这么多，核心还是想说明大数据两大核心为云技术和 BI，离开云技术，大数据就没有根基和落地的可能；离开 BI 和价值，大数据又变为舍本逐末，丢弃关键目标。简单总结：大数据目标驱动是 BI，大数据实施落地是云技术。

大数据的本质就是利用计算机集群来处理大批量的数据，大数据的技术关注点在于如何将数据分发给不同的计算机进行存储和处理。

云计算的本质就是将计算能力作为一种较小颗粒度的服务提供给用户，按需使用和付费，体现了以下几个特点。

（1）经济性。经济性是指不需要购买整个服务器。

（2）快捷性。快捷性是指即刻使用，不需要长时间的购买和安装部署。

（3）弹性。弹性是指随着业务增长可以购买更多的计算资源，可以需要时购买几十台服务器的 1 个小时时间，运算完成即可释放。

（4）自动化。自动化是指不需要通过人来完成资源的分配和部署，通过 API 可以自动创建云主机等服务。

云计算的技术关注点是如何在一套软硬件环境中，为不同的用户提供服务，使得不同的用户彼此不可见，并进行资源隔离，保障每个用户的服务质量。

2.5 大数据产业

数据作为一个新兴的产业，一直处于舆论的风口浪尖。

当前有人认为大数据产业“过热”，事实上，大数据的潜力应该还远远没被释放出来。目前从实际情况来讲，大数据产业在硬件支持方面的布局如火如荼，大数据中心也在纷纷建设之中，但是数据与数据的碰撞、数据价值的开发、数据价值的应用落地等，仍然处于初期阶段。随着借助传感器、可穿戴设备、智能感知、视频采集、增强现实等技术可实现实时的信息采集和分析，这些数据能够支撑智慧城市、智慧交通、智慧能源、智慧医疗和智慧环保的理念需要。大数据将充分实现对交通、医疗、教育、环保、农业等产业的升级改造，以及大数据应在社会生产、生活更广的范围内得到普及。

大数据的价值体现在预警、预测、决策、智能等方面。预警，即通过数据采集、数据挖掘、数据分析，对已经存在的风险发出预报与警示；这个体现在气象预测、地震预测以及各个行业的舆情分析。预测，是指立足于纵向时间轴，对相对长时间内某些问题的判断形成指导；这个很多应用在金融行业，比如当前很热的量化投资。决策，是指通过所有相关数据的联动，形成基于数据和分析之上的决策或结论；智能，即当我们基于对现实问题的分析与判断，通过技术手段实现智能化的行为。大数据相关的科技公司解决的问题基本上也是从这几方面进行发挥的。

当然，大数据也面临着一些需要逐步解决的问题，包括技术问题、数据孤岛问题等。比如政府部门存在“不愿开放、不敢开放、不会开放”数据的问题，如何开放共享仍有待探索。

2.6 大数据人才

传统的数据挖掘工具和BI系统已经存在很久了，通过各类报表展示，让管理层了解企业运营信息，过去的确帮助企业提高过管理水平，达到了预期目的。

在大数据时代，企业需要的是实时数据，需要的是高效工具，需要的是决策支持和预测。传统的数据挖掘工具的性能和灵活性已经不能满足企业的需要，另外非机构化数据的应用也对传统数据工具提出了挑战。BI领域中的SAS、SPSS、TD等数据工具越来越被边缘化，R和Python语言正在成为数据统计和可视化的新宠。

数据的时间价值正在得到重视，特别是金融企业，所有的业务部门都期望在最短的时间里，看到资金使用情况、客户交易情况以及风险管控情况。企业越早了解信息，就会越早进行决策，时间就是金钱。过去数据需求可能是(T+5)或者(T+30)，现在的数据需求往往是(T+1)或者(T+0)，数据实时性、准确性、相关度被提到了一个非常重要的地位。业务的需求已经很明显了，但是数据工具和人才却是一个很大的挑战。

中国200多家大数据企业，看到了大数据产业的曙光，看到了大数据产业的价值，同时也在经历着大数据企业的痛苦。大数据产业发展很快，市场正在逐步变大，但是其产业优势不明显，优势企业很少，数据商业化较慢，市场还不成熟，客户数据商业敏感度较低，缺乏高质量数据工具和人才。所有大数据企业内心的感受就是，站在了时代的风口，选对了方向和行业，但是想发展壮大还是很难。200多家大数据企业正在努力耕耘着大数据产业，痛并快乐着。

就数学和统计专业而言，目前绝大多数的数据科学家有数学和统计专业背景，所以你选择的学校可能并没有所谓“数据分析”这个专业，但是其数学和统计专业很可能有开设一系列课程帮助你培养大数据分析的能力，甚至还会建议你去选修一些外系的编程或市场营销课程来丰富你的技能组合。

另一个进入大数据领域的方式是学习计算机科学专业，这一路线将会侧重于学习大数据采集和分析的技术问题。

目前市面上许多的大数据技术，如MapReduce、NoSQL和Hive就是来源于软件工程师的发明创造。所以如果你对计算机科学感兴趣，又想在毕业后从事大数据相关岗位，就可以在本科阶段侧重于对人工智能、机器学习和数据理论方面的学习。

如果说计算机科学专业的学生研究的是如何让大数据技术变得更快更好，那么商科学生学习的就是如何用大数据技术去为企业赢得利润，因此更关注的是如何把大数据技术与市场营销、产品定位和购买模式等结合起来。

与此同时，越来越多的商学院开始开设专攻商业数据分析的本科和研究生项目，尽管不像计算机科学专业那样对于理工科知识有那么高的要求，但是还会涉及一定的数据库设计、分析和编程以及相关统计软件，如Python的使用。

第 3 章 人工智能与机器学习

3.1 人工智能

这章我们介绍人工智能与机器学习。毕竟这两个词太火了,我们一样是要弄清楚它们的定义、之间的区别以及怎么通过去解释人工智能提升自己的价值。同时,还要介绍今后我们可能常会用到的一些方法,比如神经网络、贝叶斯、马尔可夫、自然语言处理等。

我们可以简单地回答说人工智能最直接的解释就是不是人的智能,但能像人那样思考,也可能超过人的智能。

人工智能(artificial intelligence,AI)被认为是 21 世纪三大尖端技术(人工智能、基因工程、纳米科学)之一,近几年来飞速发展,互联网科技巨头和无数中小创业公司投身其中,越来越多基于人工智能的应用开始渐渐走进我们的日常生活。

先举几个生活中常见的例子。

导航几乎是我们开车出行的必备应用之一,以大数据和机器学习为基础,是一种典型的地图人工智能化。在车里打开导航时,地图采集设备自动识别景物和道路特征定位你所处的位置,提取建筑轮廓并绘制形状,根据道路图形标牌、电子眼、警示牌等自动挖掘出过期或新增的信息点以及道路变化,并且根据道路实时路况计算规划出最优出行路径等。导航的整个过程不需要人工参与,机器根据算法和数据智能化输出结果,是离我们最近的人工智能应用之一。

信息获取是我们的基础需求之一,百度搜索和今日头条个性化推荐就是人工智能在信息分发领域的实际应用。经过深度学习的机器基于大数据根据你的检索关键词或者个人属性、行为记录等从数据库中自动调取、匹配和呈现信息。今日头条甚至已经有了人工智能写稿机器人,基于自然语言处理、视觉图形处理和机器学习技术等,AI 写稿机器人能够根据网络热点、评论分享、用户喜好进行文字编写、标题封面图选择等,并在两秒钟内创作一篇效果不逊于人工编辑的稿件。

AI 之所以重要,是因为它解决了极其复杂的问题,而这些问题的解决方案可以应用到对人类福祉重要的领域——从健康、教育,到商业、交通,乃至于公用事业和娱乐等。

那么,我们是不是说人工智能将取代人类或者就是灾难呢。

困难的问题是简单的,简单的问题是困难的。

其实,这里我们可以抛出一个哲学一样的话题。

2016 年 3 月注定也要载入人工智能的发展史册:来自 Google 的人工智能程序 AlphaGo 以总比分 4∶1 的成绩战胜了前世界冠军李世石。

号称“人类最后智力骄傲”的围棋也被人工智能攻破了,一时间人工智能与机器人威胁论刷爆了微博、微信及各路新闻媒体。大家都在担心着某一天自己的工作会被人工智

能抢去，又在某一天人类会被人工智能机器人统治。那场比赛中有一个细节，不知大家是否注意：这个已经在“人类最后智力骄傲”上碾压人类的AlphaGo，却连挪动一枚小小的棋子都需要人类帮助才能完成。

可能有人会说，这都不算事儿，围棋都已经战胜人类了，给AlphaGo装上机械手让它自己下棋也不过是分分钟的事儿。然而，事实真的是这么简单吗？

让计算机在智力测试或者下棋中展现出一个成年人的水平是相对容易的，但是要让计算机有如一岁小孩般的感知和行动能力却是相当困难甚至是不可能的。这便是在人工智能和机器人领域里著名的莫拉维克悖论。

莫拉维克悖论指出：和传统假设不同，对计算机而言，实现逻辑推理等人类高级智慧只需要相对很少的计算能力，而实现感知、运动等低等级智慧却需要巨大的计算资源。

这个很类似于数学发展中的理论，我们要证明最前沿的各种定理可能不难，但是要证明最简单的1+1=2可能却是最难的。

四岁小孩具有的本能——辨识人脸、举起铅笔、在房间内走动、回答问题等，事实上却是工程领域内目前为止最难解的问题。随着新一代智慧设备的出现，股票分析师、石化工程师和假释委员会都要小心他们的位置被取代，但是园丁、接待员和厨师至少10年内都不用有这种担心或者不需要担心。

3.2 机器学习

我们怎样去解释人工智能和机器学习的关系呢？

人工智能的根本在于智能——如何为机器赋予智能。而机器学习强调的是学习，或者说算法，是部署支持人工智能的计算方法。或者说人工智能是科学，机器学习是让机器变得更加智能的算法。

还记得我们强调的内容和形式的关系吗？我们可以说机器学习和人工智能是内容和形式的关系，或者说机器学习成就了人工智能。

机器学习是从大量的数据中发现规律，提取知识，并在实践中不断地完善和增强自我。机器学习是机器获取知识的根本途径，只有让计算机系统具有类似人的学习能力，才可能实现人工智能的终极目标。可以这么说，机器学习是人工智能研究的核心问题之一，也是当前人工智能研究的一个热门方向，同时也是人工智能理论研究和实际应用的主要瓶颈之一。

机器学习算法我们可以整体分为两类，不是按照有监督学习和无监督学习进行分类，而是按照容易方法论的角度进行分类的。

一种是线性化思维。体现在线性和非线性思维的转化中，比如线性回归是把非线性的转换成线性的进行研究，同时也是把高维的转换成低维的；支持向量机（SVM）其实就是反过来的，把低维的转换成高维的，把非线性的转换成线性的。

另一种是分类和聚类。在生活中，我们常常没有过多地去区分这两个概念，觉得聚类就是分类，分类也差不多就是聚类。分类与聚类之间在机器学习中有本质的区别。

1. 分类

分类是通过学习来得到样本属性与分析对象的关系。

具体来说，就是我们根据已知的一些样本，来得到分类模型(其实就是一种函数关系)，然后通过目标函数来对只包含属性的样本数据进行分类。

分类要求必须事先明确知道各个类别的信息，并且断言所有待分类项都有一个类别与之对应。但是很多时候上述条件得不到满足，尤其是在处理海量数据的时候，如果通过预处理使得数据满足分类算法的要求，则代价非常大，这时候可以考虑使用聚类算法。

2. 聚类

聚类指事先并不知道任何样本的类别标号，希望通过某种算法来把一组未知类别的样本划分成若干类别。

聚类的时候，我们并不关心某一类是什么，我们需要实现的目标只是把相似的东西聚到一起，这在机器学习中被称作无监督学习(unsupervised learning)，前面的分类就被称作有监督学习。

通常，人们根据样本间的某种距离或者相似性来定义聚类，即把相似的(或距离近的)样本聚为同一类，而把不相似的(或距离远的)样本归在其他类。

聚类的目标：组内的对象相互之间是相似的(相关的)，而不同组中的对象是不同的(不相关的)。组内的相似性越大，组间差别越大，聚类就越好。

3. 分类和聚类的关系

聚类分析是研究如何在没有训练的条件下把样本划分为若干类。

在分类中，对于目标数据库中存在哪些类是知道的，要做的就是将每一条记录分别属于哪一类标记出来。

聚类需要解决的问题是将已给定的若干无标记的模式聚集起来，使之成为有意义的聚类，聚类是在预先不知道目标数据库到底有多少类的情况下，希望将所有的记录组成不同的类或者说聚类，并且使得在这种分类情况下，以某种度量(如距离)为标准的相似性，在同一聚类之间最小化，而在不同聚类之间最大化。

与分类不同，无监督学习不依赖于预先定义的类或带类标记的训练实例，需要由聚类学习算法自动确定标记，而分类学习的实例或数据样本有类别标记。

3.3 机器学习的常用算法

机器学习的常用算法很多，而且现在有许多已实现的机器学习开源包可供我们调用。如果我们能对常见的算法熟练掌握，比如通过使用 Python 实现算法，加深对机器学习的本质理解，那么，我们在量化投资领域里面就会有更多机遇。

1. 线性回归

这里说线性回归是因为我们讲的回归基本上就是线性回归，也就是用线性的方法去解决非线性的问题。说得更极端一些，机器学习里面的所有算法都可以看作是初中学习的一次函数思想，即 $Y=AX$ 来看机器学习发展过程中的各个理论或者算法。

2. 支持向量机(SVM)

SVM 的关键在于核函数。低维空间向量集通常难于划分,解决的方法是将它们映射到高维空间。但这种办法带来的困难就是计算复杂度的增加,而核函数正好巧妙地解决了这个问题。也就是说,只要选用适当的核函数,就可以得到高维空间的分类函数。在 SVM 理论中,采用不同的核函数将导致不同的 SVM 算法。在确定了核函数之后,由于确定核函数的已知数据也存在一定的误差,考虑到推广性问题,因此引入了松弛系数以及惩罚系数两个参变量来加以校正。在确定了核函数的基础上,再经过大量对比实验等取定这两个系数,该项研究就基本完成了,适合相关学科或业务内应用,且有一定能力的推广性。

3. 聚类

所谓聚类,就是将相似的事物聚集在一起,而将不相似的事物划分到不同的类别的过程,是机器学习中十分重要的一种手段。比如古典生物学之中,人们通过物种的形貌特征将其分门别类,可以说就是一种朴素的人工聚类。如此,我们就可以将世界上纷繁复杂的信息,简化为少数方便人们理解的类别,可以说是人类认知这个世界的最基本方式之一。

在数据分析的术语之中,分类和聚类是两种技术。分类是指我们已经知道了事物的类别,需要从样品中学习分类的规则,是一种有指导学习;而聚类则是由我们来给定简单的规则,从而得到分类,是一种无指导学习。两者可以说是相反的过程。

这里,我建议读者需要熟练掌握 EM 算法。

最大期望算法(expectation-maximization algorithm,EM),又称为期望最大化算法,在统计中被用于寻找,依赖于不可观察的隐性变量的概率模型中,参数的最大似然估计。

在统计计算中,最大期望(EM)算法是在概率(probabilistic)模型中寻找参数最大似然估计或者最大后验估计的算法,其中概率模型依赖于无法观测的隐藏变量(latent variable)。最大期望经常用在机器学习和计算机视觉的数据聚类(data clustering)领域。最大期望算法经过两个步骤交替进行计算,第一步是计算期望(E),利用对隐藏变量的现有估计值,计算其最大似然估计值;第二步是最大化(M),最大化是在 E 步上求得的最大似然值来计算参数的值。M 步上找到的参数估计值被用于下一个 E 步计算中,这个过程不断交替进行。

K-means 是最为常用的聚类方法之一,其实就是 EM 算法的一种特例。

4. 分类

贝叶斯是常见的分类方法,它是在量化投资中应用最多的一种策略。

贝叶斯算法的精髓在于其提供了一种数学法则来解释当有一系列新证据出现的情况下,你该如何改变自己现有的信念。一个典型的例子就是:看到第一次日出时,接着会想知道太阳是否会再次升起。于是赋予两个可能的结果同等的先验概率,并且在一个袋子里面放入一颗白球、一颗黑球,分别代表太阳会再次升起、太阳不会再次升起。第二天,当太阳再次升起的时候,在袋子里面再放入一颗白球,于是概率由初始的 1/2,上升到了 2/3,第三天,当太阳再一次升起的时候,再放入一颗白球,此时的概率(信念的程度)已经

2. 支持向量机(SVM)

SVM 的关键在于核函数。低维空间向量集通常难于划分,解决的方法是将它们映射到高维空间。但这种办法带来的困难就是计算复杂度的增加,而核函数正好巧妙地解决了这个问题。也就是说,只要选用适当的核函数,就可以得到高维空间的分类函数。在 SVM 理论中,采用不同的核函数将导致不同的 SVM 算法。在确定了核函数之后,由于确定核函数的已知数据也存在一定的误差,考虑到推广性问题,因此引入了松弛系数以及惩罚系数两个参变量来加以校正。在确定了核函数的基础上,再经过大量对比实验等取定这两个系数,该项研究就基本完成了,适合相关学科或业务内应用,且有一定能力的推广性。

3. 聚类

所谓聚类,就是将相似的事物聚集在一起,而将不相似的事物划分到不同的类别的过程,是机器学习中十分重要的一种手段。比如古典生物学之中,人们通过物种的形貌特征将其分门别类,可以说就是一种朴素的人工聚类。如此,我们就可以将世界上纷繁复杂的信息,简化为少数方便人们理解的类别,可以说是人类认知这个世界的最基本方式之一。

在数据分析的术语之中,分类和聚类是两种技术。分类是指我们已经知道了事物的类别,需要从样品中学习分类的规则,是一种有指导学习;而聚类则是由我们来给定简单的规则,从而得到分类,是一种无指导学习。两者可以说是相反的过程。

这里,我建议读者需要熟练掌握 EM 算法。

最大期望算法(expectation-maximization algorithm,EM),又称为期望最大化算法,在统计中被用于寻找,依赖于不可观察的隐性变量的概率模型中,参数的最大似然估计。

在统计计算中,最大期望(EM)算法是在概率(probabilistic)模型中寻找参数最大似然估计或者最大后验估计的算法,其中概率模型依赖于无法观测的隐藏变量(latent variable)。最大期望经常用在机器学习和计算机视觉的数据聚类(data clustering)领域。最大期望算法经过两个步骤交替进行计算,第一步是计算期望(E),利用对隐藏变量的现有估计值,计算其最大似然估计值;第二步是最大化(M),最大化是在 E 步上求得的最大似然值来计算参数的值。M 步上找到的参数估计值被用于下一个 E 步计算中,这个过程不断交替进行。

K-means 是最为常用的聚类方法之一,其实就是 EM 算法的一种特例。

4. 分类

贝叶斯是常见的分类方法,它是在量化投资中应用最多的一种策略。

贝叶斯算法的精髓在于其提供了一种数学法则来解释当有一系列新证据出现的情况下,你该如何改变自己现有的信念。一个典型的例子就是:看到第一次日出时,接着会想知道太阳是否会再次升起。于是赋予两个可能的结果同等的先验概率,并且在一个袋子里面放入一颗白球、一颗黑球,分别代表太阳会再次升起、太阳不会再次升起。第二天,当太阳再次升起的时候,在袋子里面再放入一颗白球,于是概率由初始的 1/2,上升到了 2/3,第三天,当太阳再一次升起的时候,再放入一颗白球,此时的概率(信念的程度)已经

由 2/3 上升到了 3/4。随着时间的推移，最初认为太阳每天早晨是否升起的怀疑信念，就慢慢地被修正为几乎可以断定太阳永远会再次升起。理解了贝叶斯算法，遗传算法就容易理解了。

贝叶斯在我们理解市场微观结构，比如做市商模型或者在做高频交易策略方面，可以应用于订单流策略设计。

第4章 金融科技在金融中的应用

4.1 在量化投资中的应用

金融科技作为一种手段或者技术，可以做的事情是很多的。这里，我们只是针对有财经院校背景的比较容易接受而且很容易理解的一些应用进行讲解，对我们今后的工作也是一种期待。

对于金融科技的一种口号式的回应，我们可以说金融科技实现的可能是实现一切皆可数据化、一切皆可量化、一切皆可预测。

我们谈这些金融科技在金融中的应用也是按照一种模式来解释的，比如优势是什么，解决了什么问题，有什么不足的，等等。我们的目的也是形成一种看法，而不是听了很多概念，就像看朋友圈，信息泛滥，怎么去过滤为自己所用呢？

量化投资解决的问题是传统投资者主观投资中的不足之处。

交易前的准备阶段。在做交易决策前，可能是基于现象的预判而进行的投资，也可能是基于事件进行的投资。一个事件，一个现象，就像盲人摸象一样，看到的很多是片面的，作出的决策也许就是随机的。当然，有当年的投资者长达很多年的分析，我想这个类似自己买了一台计算机一样，多年的数据自己脑海里面一清二楚，可是我们认为这些数据远远不及现在大数据技术所体现出来的效用。

交易中的决策。这也是主观交易面临的不足，人都有情绪化或者当面临很多标的的时候，一个人或者手工就很难解决这些问题了。

量化投资就是要解决这些问题，即数据的收集和数据分析能力的提升，高速确定交易决策。

基于大数据分析的量化投资，它具有一个显著功效，就是把人的情绪排除在投资进程之外——整个投资进程完全按照人预先设定的程序进行操作。把人的情绪排除在外的一个作用就是确保投资进程的客观性，确保这样的客观性是重要的，因为量化投资策略的构建本身是基于客观的规律，这些规律由基于大数据分析总结出来的，它们具有客观的精确性。

量化投资的科学依据：“历史会重复”。

构建量化投资策略时，通常通过分析历史数据，获得经验规律，然后，把此规律用于预测市场的未来走势，以便从中获利。这里的一个科学依据是：历史往往会重复。其实，不仅股票市场过去的历史会在未来重复，同期来看，一个国家的股票市场的某些规律，也可能在另一个国家的股票市场中重复出现。

当然，这里的重复需要我们自己的理解。就像太阳每天都会升起，我们说每天都是新的还是说每天都是一样的，不同的价值观和不同的视角导致我们在对一些很简单很朴素

的道理面前理解偏差巨大，从而量化投资决策的效果差距也是很大。

量化投资的挑战或问题有两个方面，一方面量化投资是一个新兴的领域，很多方面还不完善，自然也就存在各种各样的问题，比如各种量化交易平台的搭建，关于这方面的法律的完善，等等；另一方面是因为量化只是一个手段，就像刀剑一样，对于量化的这样的技术所涉及的数据收集、数据分析以及决策也有可能助长错误的发生。

4.2 Python 在智能投顾中的应用

智能投顾并不是量化投资。

其实很多时候我们在区别很多观念或者相似词语区别时，一定先从这个概念或者词语本身去思考。量化投资强调的是通过量化的方式去做投资，量化交易也如是。智能投顾是基于人工智能的投顾。

可问题就在于，最简单的词语解释起来最麻烦。比如投顾，怎么理解投顾呢？业内有不同的说法，有一些把股票操盘手叫作投顾，有一些把证券公司的客户经理叫作投顾，有一些把私募基金的基金经理叫作投顾。

投资顾问其实就是解决用户和金融产品端的沟通。具体的说法是，投资顾问服务于把金融产品更优化地面向用户进行配置。

智能投顾就是通过人工智能来更好地实现这样的服务。

很多纷繁复杂的信息、概念，我们用学习数学的方式来建立模型，理解和表达更容易，更好地提升自己的价值。

那么智能投顾解决的是哪些问题呢？

一方面是根据用户不同的风险需求定制资产组合。这个就像厨师做的工作一样，厨师有各种食材，根据客户的口味做成客户最期待的，给客户最优化的体验。智能投顾就是通过大规模的金融计算，结合金融投资组合理论计算最优投资组合，让资产配置最优化，让投资决策更理性。

另一方面是怎样去量化客户的风险需求以及其他需求，因为沟通成本巨大，可能很多时候我们和客户交流一天或者一周都不大清楚客户到底需要什么，或者客户都不清楚自己的需求。但是我们通过大数据，比如客户的消费记录、信用记录，以及客户可以查询到的公开信息能够获取客户的风险偏好。

当然，我们可以更加模型化地概括智能投顾的一个特点。

第一步是通过大数据技术获取客户个性化的风险需求以及变化规律。

第二步是通过各种算法定制个性化的资产配置解决方案。

第三步是对这个解决方案进行实时跟踪并能作出相应的调整。

我们会发现很多东西在“三”以内进行解释是最容易被理解的。我们向对方解释不用谈太多，三点以内；要塑造价值通过各种灵活的方式做二分法。

4.3 Python 在征信中的应用

在金融行业工作中，征信应该是被理解得最深刻的一个词了。

甚至，我们可以说做金融就是做征信。

征信和我们日常生活中的沟通、购物一样都是在收集我们的数据，最核心的是这些数据直接和我们个人融资能力是相关的。

长久以来，QQ 和淘宝垄断着我们的线上沟通工具和购物，同样征信却一直是央行的权利。随着大数据技术的发展，腾讯也有微粒贷，阿里巴巴有芝麻信用。他们都在做着征信的工作。

中国有着十多亿人口，没有征信记录的有五亿多人，怎么满足这些人的金融需求，这就是做大数据征信的市场巨大机遇。

说了这么多，我们还是按照同样的方法论来解释征信。

征信的定义或者本质是什么呢?

收集用户的数据对用户本身的特征描绘和风险判断有显著作用就是征信。

这句话有点长，简单地说，能成为信用数据的就是征信。

事实也是这样，现在获取信息的渠道和能力都越来越强了，一切信息皆可以成为信用数据，经过分析后都可用于证明一个人或企业的信用状况。

因为数据覆盖广、维度多，因此形成了广义的征信，我们也可以叫作大数据征信。

大数据征信主要还是数据问题，怎么获取数据、清洗数据，事实上这些需要前期投入巨大的人力、物力，不仅周期长且回报慢。尤其是个人征信这部分，对于数据、资金、技术，以及场景都有很高的要求。个人征信公司经营成功的关键可能就在于：数据来源的范围和准确性、数据处理能力、数据产品是否能够满足客户要求、是否具有多样性。

从大数据征信的解决的问题来看，一方面确实是解决个人的金融需求，比如贷款等；另一方面大数据征信已从金融业务向生活服务蔓延。其中，最核心的两个价值就是：防范欺诈风险和信用风险。简单来说就是：既要证明“你是你”，还要描述出“你是什么样的人”。现在，人脸识别因其技术的成熟度和准确率，以及使用的便捷性而被进一步普及。腾讯征信、芝麻征信以及很多银行贷款都在使用这种技术，在一般情况下，只需要拿着手机拍照，银行系统就会自动获取你的信息并快速告诉你的信用评级和各种支持的金融服务。

大数据征信还面临的问题或者不足的怎么去看呢，其实和量化投资一样，大数据征信也是一种技术手段，不可避免会遇到这些问题。这个读者可能有体验，比如申请各个银行的信用卡，授权给你的额度每个银行都是不一样的，除了获取的数据也许不一样之外，更大的不同是各个银行使用的信用风险模型不一样，这就会导致你得到的信用卡额度不一样。当然这也许就是每个银行在大数据征信的核心竞争力。

第 5 章 Python 在金融行业的应用

每个人的核心竞争力就像生活中每天刷信用卡一样，是逐渐累积的。

5.1 Python 在金融行业的现状和应用

1. 编程和金融

几年以前，可能财经院校的毕业生，而且更多的是研究生，在校期间可能想到的是毕业后进银行，其实现在去银行的很多时候也是从柜台做起，不是不优秀，是资源配置所导致的，我想大家也是知道的。可现在是，银行柜台可能都会逐渐减少或者消失，很多营业网点柜台人员也是越来越少，办业务的人也越来越少的。在“90 后”全面接管的消费市场中，通过手机很多业务都是可以办理的。拿着手机拍个照，个人的信用马上就会被银行读取，各种业务权限和信用等级，银行系统都能及时显示。

科技终是在改变生活。

大数据、云计算以及这两年的区块链这些词语在金融行业中使用是最频繁的，不在朋友圈发几篇相关的文章，嘴里不说这些词语感觉就不是玩金融的一样。当然，事实也是科技在金融中的应用是最前沿，最全面，影响也是最大的。

对于金融行业，今后几十年都会被看作是年轻人渴望从事的行业，而且毫无疑问是朝阳行业，一直也会是朝阳行业。对于渴望在这个行业能够获取超额回报的，自身也是需要具有核心竞争力的。

对于就业来说，我们可以将金融行业分为金融机构和金融科技类企业。这里的金融机构包括银行、证券、基金、信托、保险、期货等，金融科技包括互联网金融企业、金融大数据企业和量化投资平台。互联网金融企业主要包括 P2P，金融大数据包含消费金融、监管科技等；量化投资平台包括像京东金融这样的企业。

对于金融机构来讲，我们也是要么拼学历和证书，要么拼工作经验。毕竟这些东西都是客观的、形式的，也正因为这样才是可以量化的、评估的，比如你是 985 还是 211，你是研究生还是本科生，你是在中国银行工作的还是在某一家跑路了的公司待过的，你是工作了 10 年还是工作了 3 年，你是考了 CFA 二级还是考了基金从业资格。如果你说你会这个信用风险建模的某个厉害的算法，或者你说你会用 BS 公式的某一个结论得出某个类型的期权定价数值计算公式，应聘的时候也是会碰壁的，毕竟很难量化很难评估你是万里挑一的人才，就看其他人是怎么理解的了，即使你自己后来单独创业获得了 A 轮融资。

不是名校，又没有证书，就没有可以拼的吗？

对于金融科技企业来讲，我们可以去应聘吗？当然可以的，不管是前台、中台还是后台，因为金融科技，服务的也是金融类机构或者也是和金融相关的。那我们凭什么去金融

科技企业呢?

“在未来,很可能一群把 Python 语言玩弄在股掌之间的人,将成为金融界的新星。”

就跟 Javascript 在 Web 领域无可撼动的地位一样,Python 也已经在金融量化投资领域占据了重要位置,从各个业务链条都能找到相应的框架实现。

金融科技企业大部分都是用 Python,后台用 Python 和 C++ 做大数据系统构建,中台用 Python 做数据分析,展示给客户的也用 Python。

Python 是一门比较全面与平衡的语言,既能满足包括 Web 在内的系统应用的开发,又能满足数据统计分析等数学领域的计算需求,同时也能作为胶水语言跟其他开发语言互通融合。

Python 在金融行业简直就是主流语言。

2. 人生苦短,我用 Python

(1) Python 介绍。你可能已经听说过很多种流行的编程语言,比如非常难学的 C 语言,非常流行的 Java 语言。

那 Python 是一种什么语言呢?

比如,完成同一个任务,C 语言要写 1 000 行代码,Java 只需要写 100 行,而 Python 可能只要 20 行。

如果你是来自财经院校或者不接触编程好多年了,比如你会使用计算机,但从来没写过程序;还记得初中数学学的方程式和一点点代数知识;想从编程小白变成专业的金融大数据架构师;每天能抽出半个小时学习。那么这本书是适合你的。

(2) Python 的特点。Python 有如下特征。

① 开源。Python 和大部分可用的支持库及工具都是开源的,通常使用相当灵活和开放的许可证。

② 解释型。CPython 参考实现是该语言的一个解释程序,在运行时将 Python 代码翻译为可执行字节代码。

③ 多用途。Python 可以用于快速、交互式代码开发,也可以用于构建大型应用程序;它可以用于低级系统操作,也可以承担高级分析任务。

④ 跨平台。Python 可用于大部分重要的操作系统,如 Windows、Linux 和 Mac OS;它用于构建桌面应用和 Web 应用;可以在最大的群集和最强大的服务器上使用,也可以在树莓派(http://www.raspberrypi.org)这样的小设备上运行。

⑤ 动态类型。Python 中的类型通常在运行时推知,而不像大部分编译语言那样静态声明。

(3) Python 和 MATLAB/R 语言。做数据分析、科学计算等离不开工具、语言的使用,目前最流行的数据语言,无非是 MATLAB、R 语言和 Python 这三种语言。为什么 Python 比 MATLAB、R 语言好呢?

其实,这三种语言都有很多数据分析师在用,但更推荐 Python,主要是有以下几点理由。

① Python 易学、易读、易维护,处理速度也比 R 语言要快。

② Python 势头猛,众多大公司需要,市场前景广阔;而 MATLAB 语言比较局限,专

注于工程和科学计算方面。

③ Python 具有丰富的扩展库，这个是 MATLAB 和 R 语言所不能比拟的。

(4) Python 版本选择。初学者版本选择是很多人都会问的，我们也不会偏向某个版本，主要根据自己的所需，毕竟只是一个编程的工具，我们面向的对象不是编程高手。本书建议选择最新版本 Python 3.x，而且本书的案例实现都是基于 Python 3.x。

至于为什么选择 Python 3.x，这是关于 Python 的一个最具争议的话题。你可能总是不能避免地遇到，尤其是如果你是一个初学者，并且一定要给个理由，我们可以有以下两种看法。

① 更整齐和更快。Python 开发者修正了一些固有的问题和小缺点，以此为未来建立一个强大的基础。这些可能不是很相关的，但最终会很重要。

② 这是未来。2.7 是 2.x 族发布的最后一个版本，并且最终每个人都要转移到 3.x 版本。Python 3 是在过去 5 年已经发布过的稳定版本，并将继续。

当然，不管怎样，读者应该专注于学习 Python 语言。版本之间的转换应该只是一个实际问题。毕竟，对我们来说，都只是一种工具。

3. Python 在金融中的应用

(1) Python 用于金融中的数据分析。在金融行业，我们可以初步把 Python 的应用分为数据分析、量化投资和金融大数据平台搭建。

数据分析可以使用 Python 实现，有足够的 Python 库来支持数据分析。Pandas 是一个很好的数据分析工具，因为它的工具和结构很容易被用户掌握。对于大数据来说，它无疑是一个最合适的选择。即使是在数据科学领域中，Python 也因为它的“开发人员友好性”而使其他语言相形见绌。一个数据科学家熟悉 Python 的可能性要比熟悉其他语言的可能性高得多。

除了 Python 在数据分析中那些很明显的优点(易学、大量的在线社区等)之外，在数据科学中的广泛使用，以及我们今天看到的大多数基于网络的分析，是 Python 在数据分析领域得以广泛传播的主要原因。

金融中的数据分析包括在银行、信托或者证券等方面的应用，比如读者在就业工作中不管是在银行或者基金都会面临数据分析，比如对期权定价方面的各种计算、波动率的计算、风险测度 VaR 的计算以及各种投资组合的优化等都是作为数据分析需要用到的。

在数据分析方面，没有其他语言能像 Python 这样既能精于计算又能保持性能，对于时间序列数据的处理展现了简单便捷的优势。

(2) Python 用于量化投资。现在量化投资在国内是很热的了，毕竟股指期货以及今年的豆粕期权等商品期权上市带来了更多的投资机遇，而对于把握好这些投资机遇的一个方面就是做数据分析工作。

对于数据的支持方面，国内有 Wind 和 Choice；对于量化交易平台有通联、聚宽和米宽等。对于这些平台，我们将在第三章做介绍。

3. Python 用于金融大数据平台搭建

在金融大数据方面，比如作为征信和金融大数据的必备技术爬虫，读者感兴趣可以去学习，因为很多网络数据自己也是可以写程序进行收集和处理的，最后还需要学习

Hadoop 系统的生态。

5.2 青春不老，奋斗不止

1. 动机

动机很重要，读者学习 Python 的动机是首要的。比如如果只是希望做做数据分析，可以专门学好 Pandas；如果希望做私募基金或者量化投资，不仅除了熟练使用 Pandas 外，还需要多学习各种估值框架、各种策略的 Python 实现以及绘图和可视化；如果编程真的是成为兴趣所在，还可以做金融大数据架构，这时候你就可以学习 Spark 以及 Web 集成等更专业的技术了。

2. 方法论

这里的方法论和读者看到这本书所体现的思想是一致的，一方面对于知识的分类，其方法很重要；另一方面对于工具的掌握是模型主义，其实这也是做项目的一种思路，我们需要先做的就是建立一个模型，再把这个模型掌握好，有了这么一个框架，然后逐渐详细、深入、补充和完善。

3. 一本书主义

比如，你拿到的这本书，也许不到两个小时就翻完了，但是我希望你按照上面的进行练习，大概知道怎么去抓取数据以及怎么数据分析，我想接下来你会很快掌握这门语言的，毕竟 Python 仅仅是一种工具。

第 6 章

Python 的环境搭建

6.1 Anaconda 介绍

1. 为什么选择 Anaconda

对于初学者,Python 的安装也许就是个头疼的事情,尤其是面临 Python 版本的困境,浪费了很多宝贵的时间,甚至让人有一种想放弃的感觉,所以今天我们介绍一种简便的安装方法,可以完美地兼容 Python 2.7 和 Python 3.5,并集成了许多包,免去配置环境变量的烦恼,让大家舒舒服服地学习 Python,少走弯路,并迅速掌握一门技能。

我们选择 Anaconda 的优势有如下几点。

(1) 不需要重新安装 Python,直接在 Windows 环境下一直单击确定就能搞定。

(2) 集成很多包。对于从事金融行业的人员来说,需要用的包都基本上已经安装了,不需要再一个一个地去下载安装。接下来我们介绍安装国内需要的 TuShare 安装包,其在 TuShare 官网也有所介绍。

(3) 包的安装很简单,Conda 可一键解决。

2. 详细的安装步骤

(1) 下载 Anaconda,其网址为 https://www.continuum.io/downloads。

① 如果国外的网站下载不了,或者读者感觉下载速度很慢的话,可以用清华镜像下载:http://mirrors.tuna.tsinghua.edu.cn/help/anaconda/.

② 如果你的计算机是 64 位就选择 64 位,32 位就选择 32 位即可。

(2) 然后一路安装就可以了,自己可以选择安装盘。安装完之后有如图 6-1 所示的几个程序。

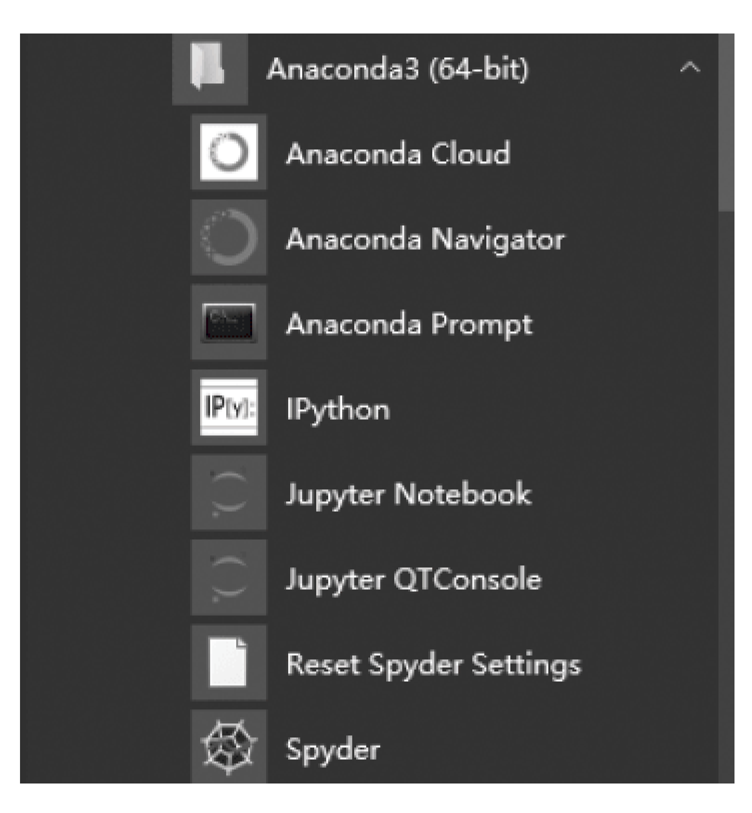


图 6-1

3. 使用 Anaconda Prompt

Anaconda Prompt 是 Anaconda 的管理器，读者可以依次单击：开始→所有程序→Anaconda，然后单击 Anaconda Prompt，如图 6-2 所示。

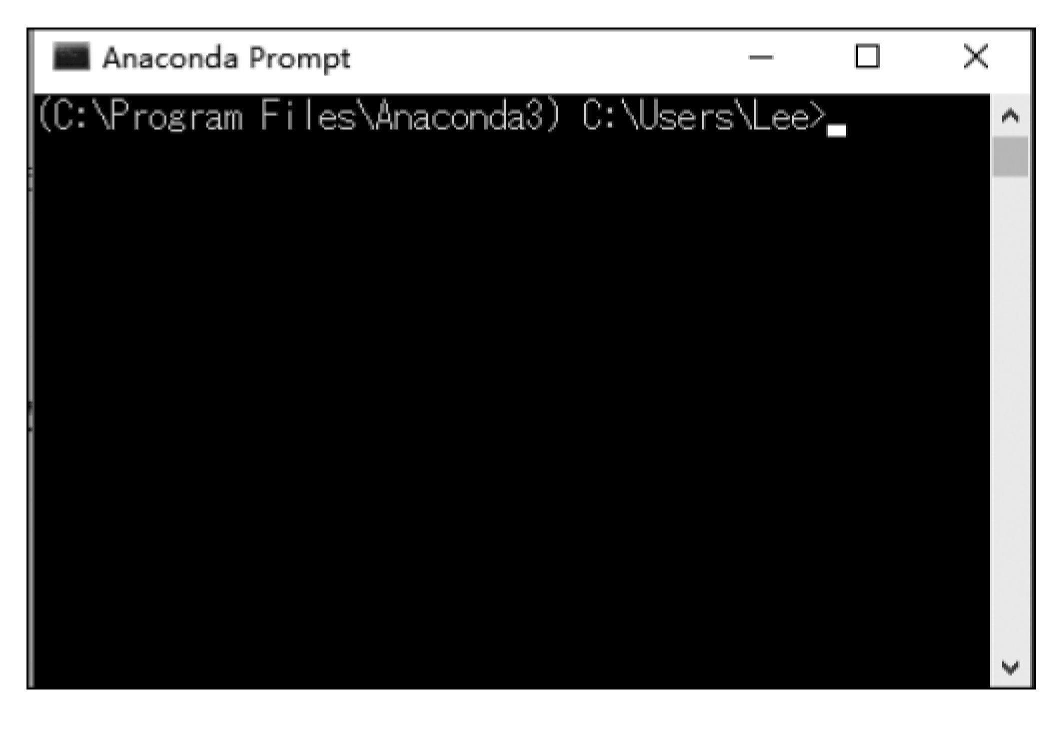


图 6-2

(1) 我们可以在 Anaconda command prompt 窗口中输入指令 conda list，就可以看到安装时自带的 Python 扩展，如图 6-3 所示。

```
Anaconda Prompt

(C:\Program Files\Anaconda3) C:\Users\Lee>conda list
# packages in environment at C:\Program Files\Anaconda3:
#
_license                  1.1                    py35_1
_nb_ext_conf              0.3.0                  py35_0
alabaster                 0.7.9                  py35_0
anaconda                  4.2.0             np111py35_0
anaconda-clean            1.0.0                  py35_0
anaconda-client           1.5.1                  py35_0
anaconda-navigator        1.3.1                  py35_0
argcomplete               1.0.0                  py35_1
astroid                   1.4.7                  py35_0
astropy                   1.2.1             np111py35_0
attrs                     15.2.0                 py35_0
attrs                     16.3.0                  <pip>
babel                     2.3.4                  py35_0
backports                 1.0                    py35_0
beautifulsoup4            4.5.1                  py35_0
bitarray                  0.8.1                  py35_1
blaze                     0.10.1                 py35_0
bokeh                     0.12.2                 py35_0
搜狗拼音输入法 全 :        2.42.0                 py35_0
bottleneck                1.1.0             np111py35_0
```

图 6-3

很明显在图 6-3 中，我们看到 Anaconda 集成的强大安装包，涉及计算及图片处理，还有机器学习，对于金融从业人员来讲，这些安装包足够我们使用了。

(2) 使用 Conda 来安装和卸载安装包。同样是在此窗口中，分别输入 conda install+安装包的名字。这里要注意的是单词之间有空格，然后回车。一般情况下就是不断输入“y”或者按 Enter 键就可以了。

比如，安装 NumPy 包，输入的命令就是：

```
conda install numpy
```

单词之间空一格，然后回车，输入“y”就可以了。

(3) 如果读者用 Conda install 安装包安装 name 没有成功，可以直接打开命令终端，用 pip 命令，比如你需要安装 TuShare，就可以使用 Pip install TuShare。如图 6-4 所示。



图 6-4

因为我的计算机上已经安装了 TuShare 包，所以会显示图 6-4。

为什么用命令终端安装呢，因为这样最方便。

4. 使用 Python

读者可以直接在 Anaconda Prompt 中输入 Python(或者依次单击：开始→所有程序→Anaconda，然后单击 Python)。如图 6-5 所示。

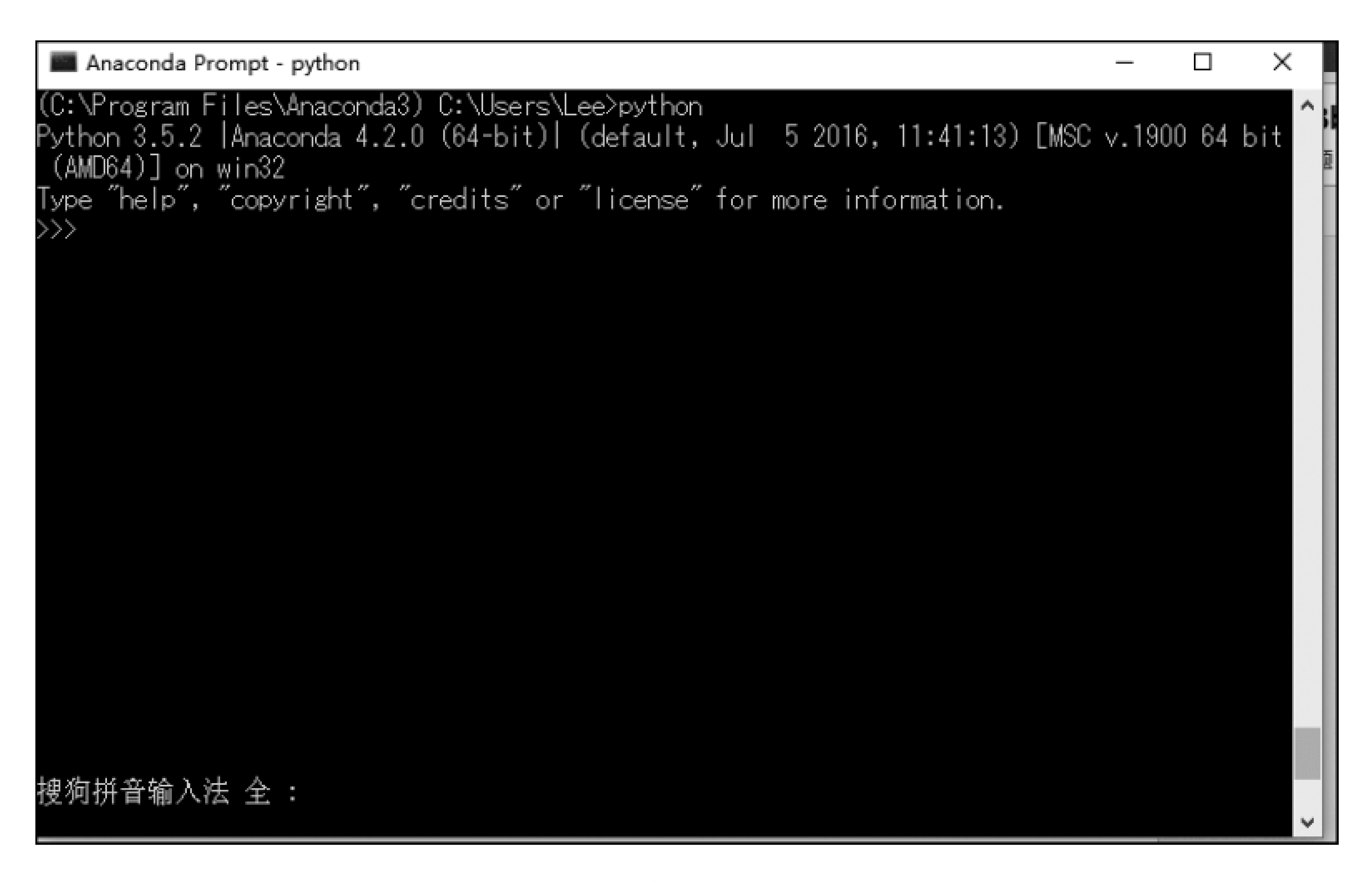


图 6-5

当读者看到有三个连续的“＞”就进入到 Python 程序中了。

比如我们简单互动一下：输入 3＋2，如图 6-6 所示。

然后你就可以看到结果为 5，这个例子只是想表明，Python 相对简单直接，很好入门，很直观，不像学习 C/C++ 语言那样需要各种配置以及头文件等，写上好几行程序才看

```
Anaconda Prompt - python
(C:\Program Files\Anaconda3) C:\Users\Lee>python
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64 bit
 (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+2
5
>>> _

搜狗拼音输入法 全 ：
```

图 6-6

到你要看到的结果。

当然，你也可以直接在命令终端中输入 Python，启动 Python 程序，效果是一样的。

如果需要退出 Python 程序，输入 exit()就可以了等，如图 6-7 所示。

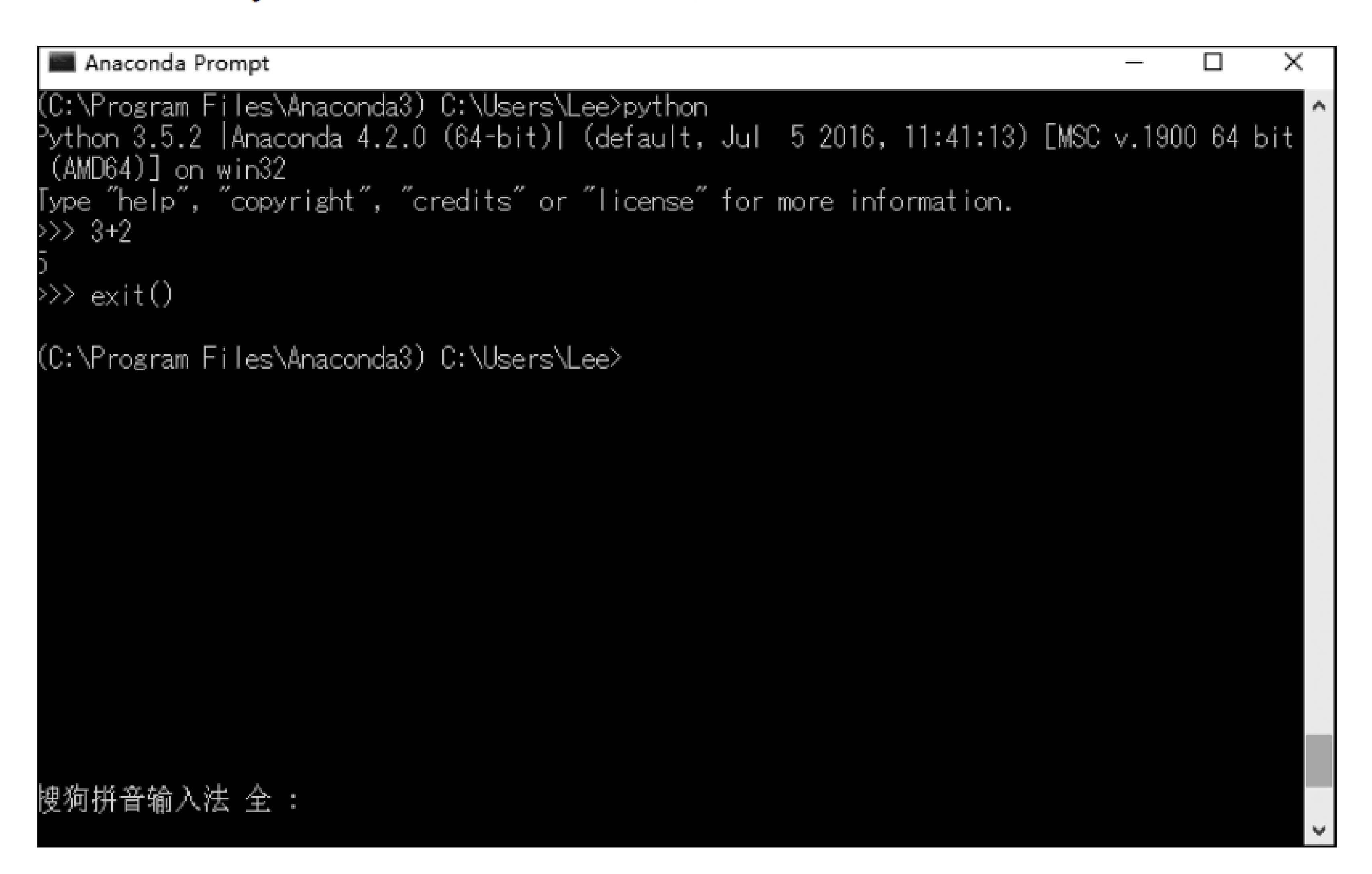


图 6-7

读者可能会发现，这样的互动还是不那么友好，那么还有更友好的界面，那就是 IPython。

5. 使用 IPython

与 Python 相同，读者在 Anaconda Prompt 中输入 IPython（或者依次单击：开始→所有程序→Anaconda，然后单击 IPython）。如图 6-8 所示。

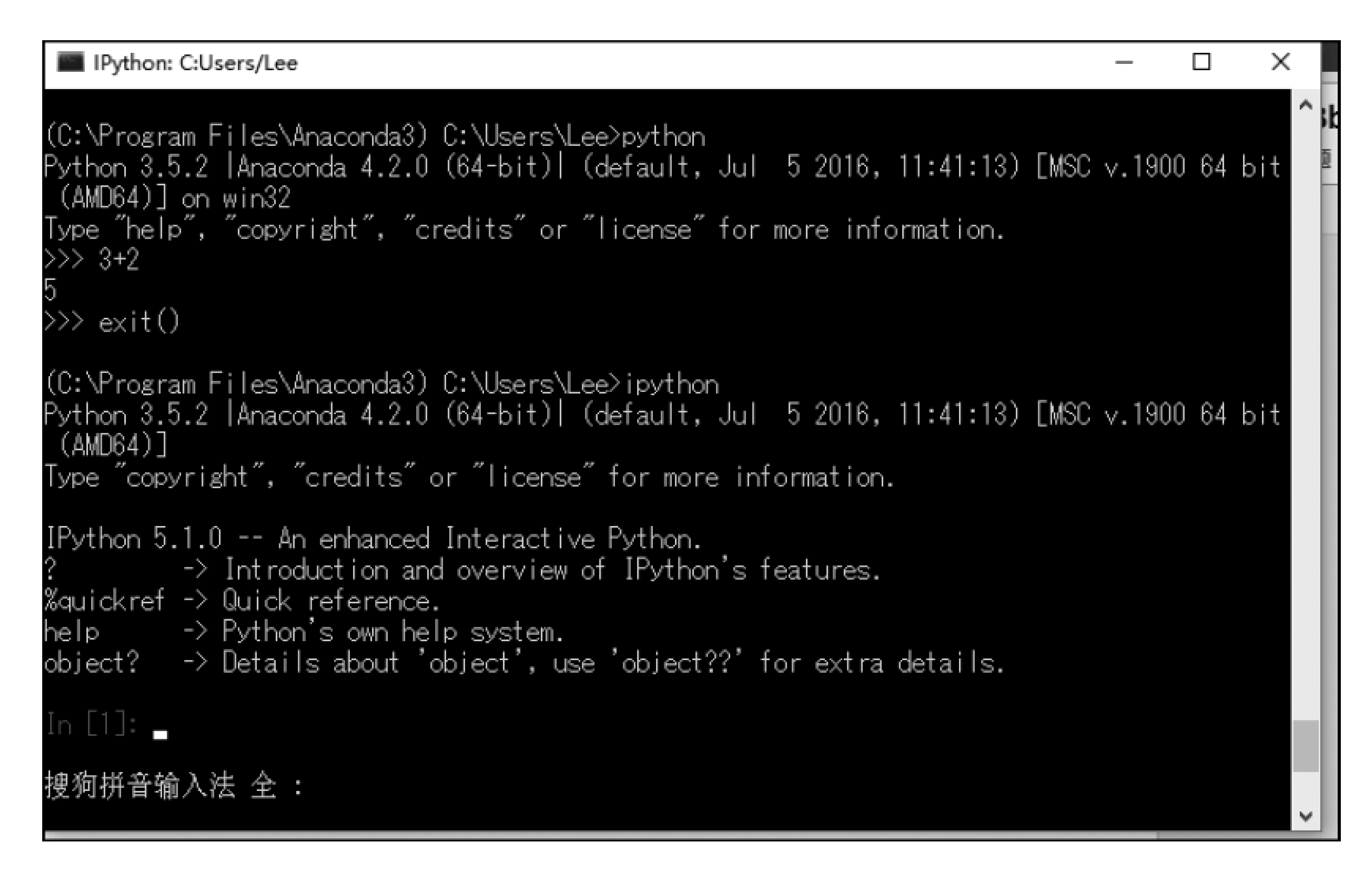


图 6-8

我们同样输入：3+2，然后按 Enter 键，如图 6-9 所示。

```
IPython: C:Users/Lee
 (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+2
5
>>> exit()

(C:\Program Files\Anaconda3) C:\Users\Lee>ipython
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64 bit
 (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: 3+2
Out[1]: 5

In [2]:

搜狗拼音输入法 全 :
```

图 6-9

读者可以看到，这里有提示 In 和 Out，就是输入和输出。比如接下来又使用求绝对值的函数 abs，直接输入 abs(99)，如图 6-10 所示。

读者如果使用过 Excel、MATLAB 或者其他计量软件，就会发现这个交互性非常好。

6. 使用 Spyder

一直使用这样的界面可能读者还是觉得不大习惯，那么我们接下来就学习一下 Spyder。

Spyder(scientific python development environment)是一个强大的交互式 Python 语

```
IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: 3+2
Out[1]: 5

In [2]: abs(99)
Out[2]: 99

In [3]: abs(-99)
Out[3]: 99

In [4]:

搜狗拼音输入法 全 :
```

图　6-10

言开发环境，提供高级的代码编辑、交互测试、调试等特性，支持包括 Windows、Linux 和 OS X 系统。

读者可以直接在 Anaconda Prompt 中输入 spyder（或者依次单击：开始→所有程序→Anaconda，然后单击 Spyder）。启动的时候可能会感觉有一点慢，稍微等一下就可以了，如图 6-11 所示。

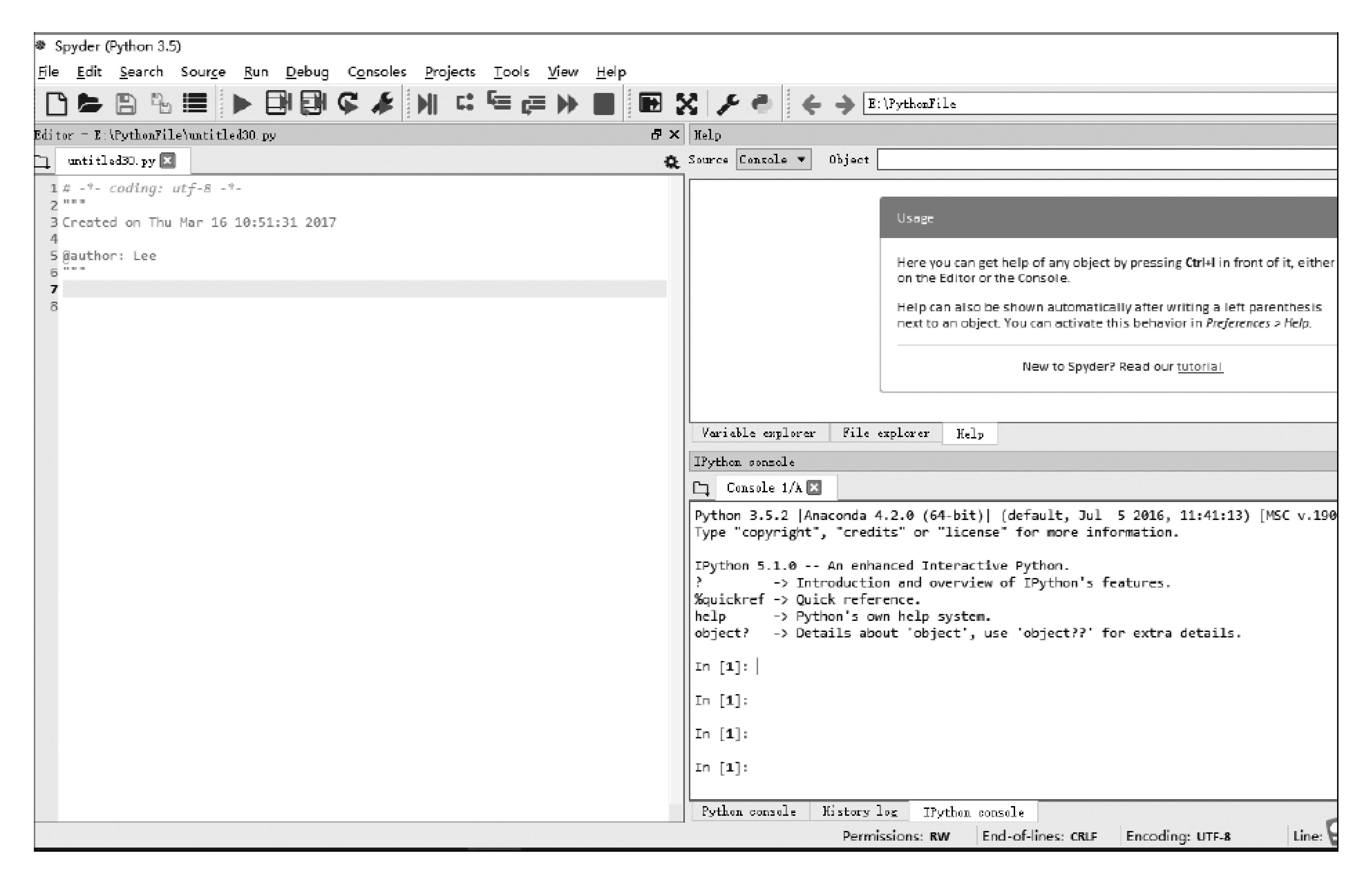


图　6-11

可以看到 Spyder 的界面设计和 MATLAB 十分地相似，熟悉 MATLAB 的读者可以很快地习惯使用 Spyder。

7. 使用 Notebook

Jupyter Notebook 目前已经成为用 Python 做教学、计算、科研的一个重要工具。本

文介绍 Jupyter Notebook 的一些基本用法。

Jupyter Notebook 使用浏览器作为界面，向后台的 IPython 服务器发送请求，并显示结果。

读者可以直接在 Anaconda Prompt 中输入 jupyter notebook(或者依次单击：开始→所有程序→Anaconda，然后单击 jupyter notebook)，如图 6-12 所示。

```
(C:\Program Files\Anaconda3) C:\Users\Lee>jupyter notebook
[I 18:48:29.743 NotebookApp] [nb_conda_kernels] enabled, 2 kernels found
[I 18:48:31.571 NotebookApp] \u2713 nbpresent HTML export ENABLED
[W 18:48:31.583 NotebookApp] \u2717 nbpresent PDF export DISABLED: No module named 'nbbrowserpdf'
[I 18:48:32.755 NotebookApp] [nb_anacondacloud] enabled
[I 18:48:32.804 NotebookApp] [nb_conda] enabled
[I 18:48:33.144 NotebookApp] Serving notebooks from local directory: C:\Users\Lee
[I 18:48:33.145 NotebookApp] 0 active kernels
[I 18:48:33.145 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/
[I 18:48:33.146 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

图 6-12

同时，浏览器打开效果如图 6-13 所示。

图 6-13

然后，单击右边 New 下拉框，选中 Python [conda root]，将显示如图 6-14 所示。

同样的，我们可以输入：3+2，然后单击上面的▶|，如图 6-15 所示。

通过上述操作，我们对 Anaconda 有了一个大概的认识，读者可以通过自行百度不断地熟练使用这些工具。

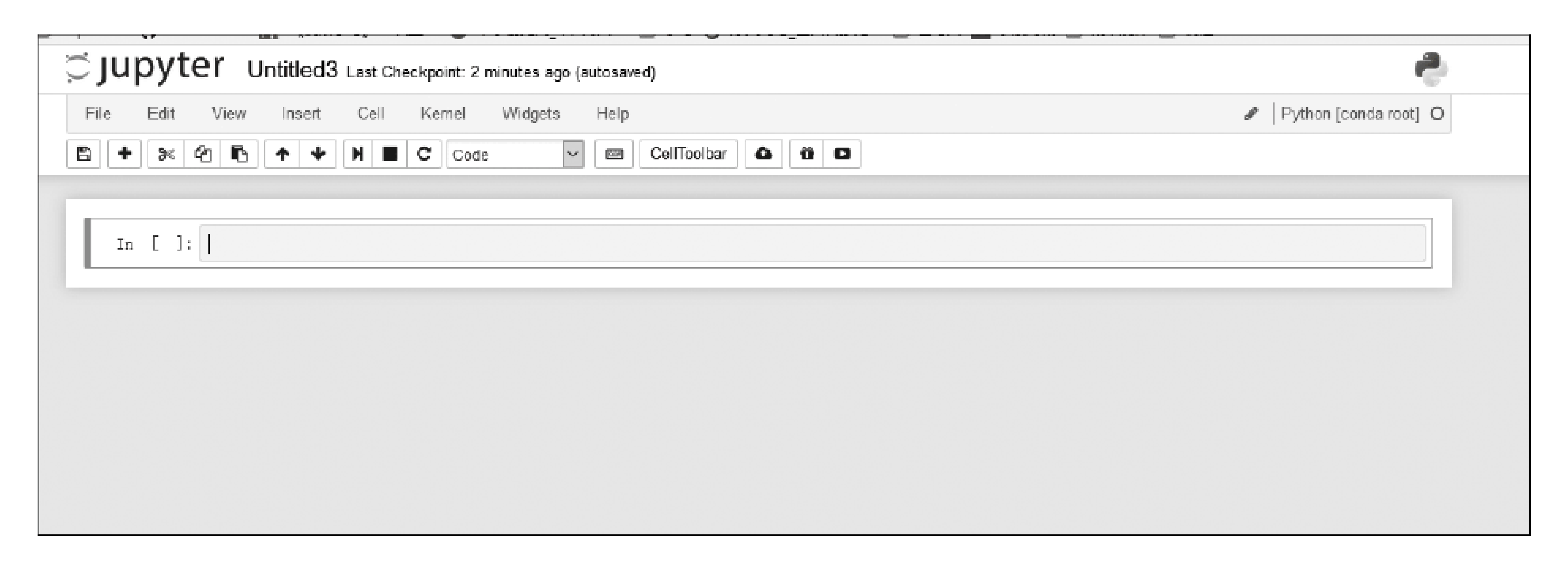


图 6-14

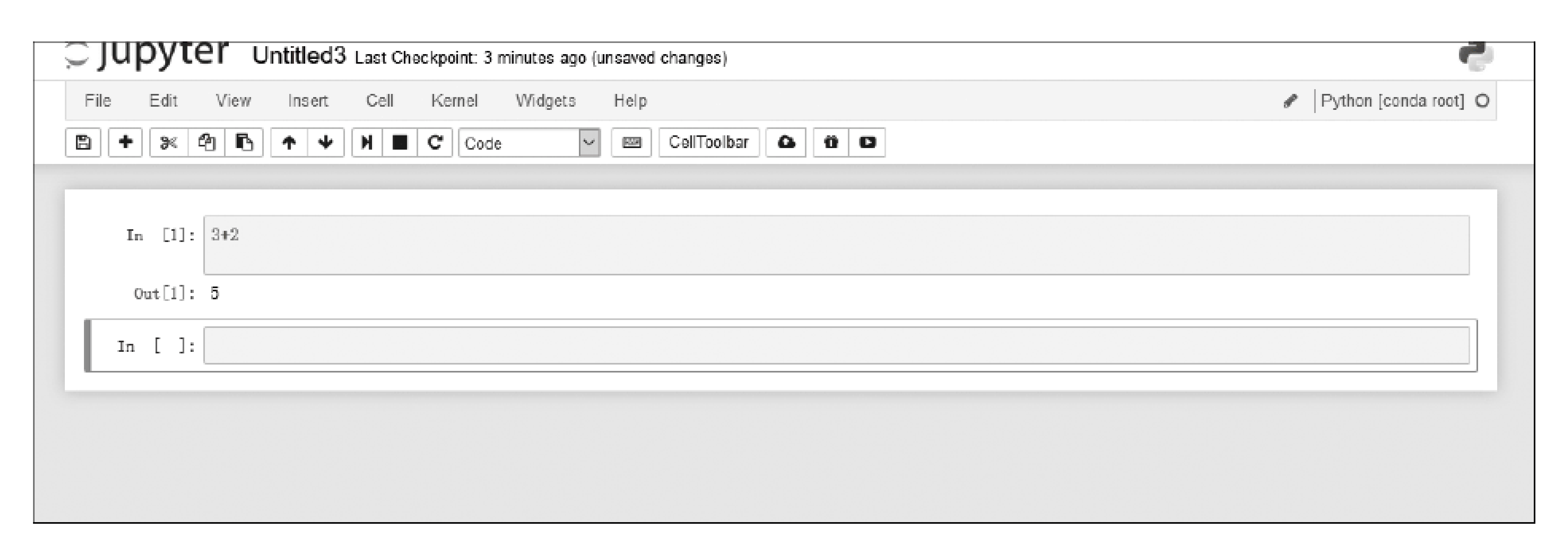


图 6-15

6.2 常见开源包介绍

这里我们针对国内读者在金融行业中的应用，介绍常见的库。

1. NumPy

NumPy(Numerical Python)是一个开源的 Python 科学计算库。使用 NumPy，就可以很自然地使用数组和矩阵。NumPy 包含很多的数学函数，涵盖线性代数运算、傅里叶变换和随机数生成等功能。

2. Pandas

Pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。Pandas 提供了大量能使我们快速便捷地处理数据的函数和方法。你很快就会发现，它是使 Python 成为强大而高效的数据分析环境的重要因素之一。

3. Matplotlib

Matplotlib 是 Python 最著名的绘图库，它提供了一整套和 MATLAB 相似的命令 API，十分适合交互式地进行制图，而且也可以方便地将它作为绘图控件，嵌入到 GUI 应用程序中。

4. TuShare

TuShare 是一个免费、开源的 Python 财经数据接口包，主要实现对股票等金融数据从数据采集、清洗加工到数据存储的过程，能够为金融分析人员提供快速、整洁和多样的便于分析的数据，为他们在数据获取方面极大地减轻工作量，使他们更加专注于策略和模型的研究与实现上。考虑到 Python Pandas 包在金融量化分析中体现出的优势，TuShare 返回的绝大部分的数据格式都是 Pandas DataFrame 类型，非常便于用 Pandas/NumPy/Matplotlib 进行数据分析和可视化。

第 7 章

Python 读取本地数据

7.1 准备知识

1. 一切都是对象

关于编程中的面向对象，就是把一组数据结构和处理它们的方法组合成一个整体。当我们理解每一个变量的时候，自然就能想到它有哪些数据结构和操作方法，最简单的类比就像一个人，具有通常的特征以及通常的做事情的方式。Python 中的任何数值、字符串、数据结构、函数、类、模块等都是对象。

2. 一些说明

(1) # 符号表示的是注释。

(2) 字符串我们用单引号(' ')或者双引号(" ")都是可以的，三引号(" " " " " ")一般用于放置文档说明(docstring)或多行字符串。比如，我们打开 Spyder 的界面，如图 7-1 所示。

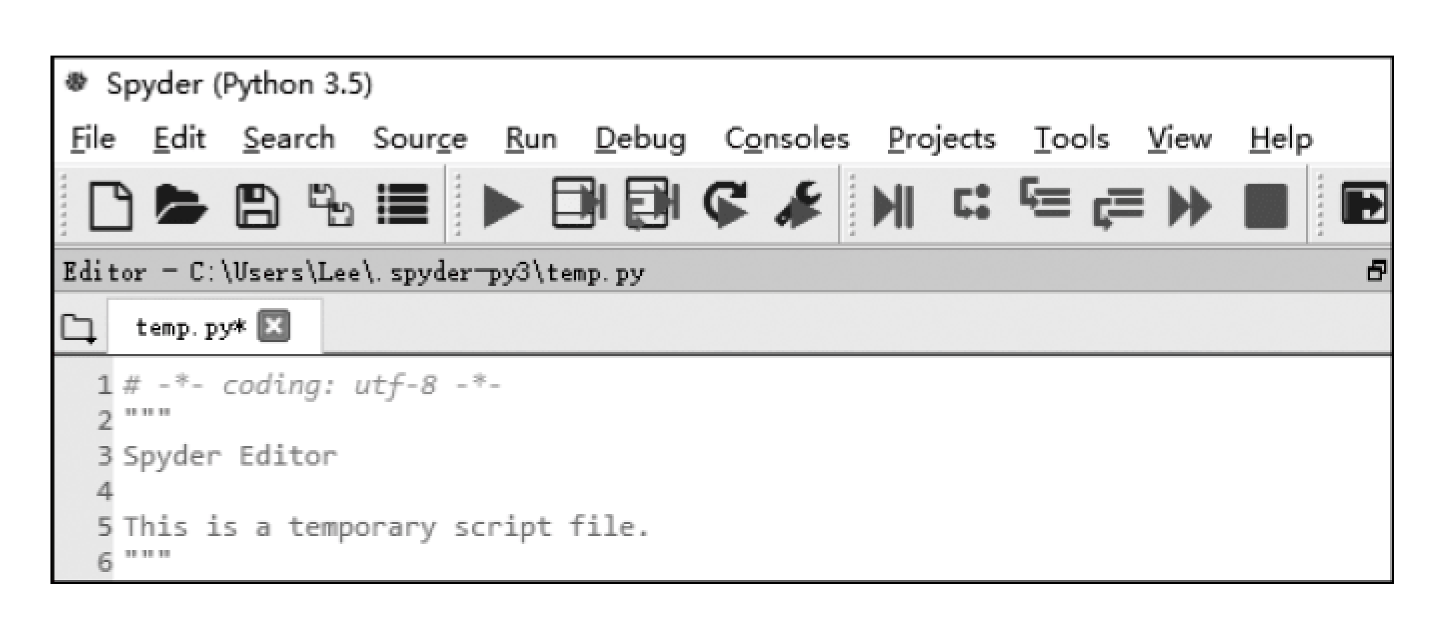


图 7-1

(3) 接下来我们演示全部都用 Jupyter Notebook，这样的安排是方便演示。读者可以直接打开 Jupyter Notebook。

(4) 我们的演示强调的是方法论，探索怎么去操作完成项目，没有很规范地解释每一个命令或函数的操作。我们的目的就是建立一个框架、一个模型，通过这样的框架或者模型快速掌握这项技能。若要让自己更加精通，那是需要不断熟练的。

7.2 本地文件的读取

常用的文件格式包括 txt、xls/xlsx、json、xml、HDF 以及其他可以转换成以上格式的数据文件。

1. 读取 txt 文件

读取文件很简单，但是我们要思路清晰。一般就是打开文件、查看数据、关闭文件，重

第 7 章

Python 读取本地数据

7.1 准备知识

1. 一切都是对象

关于编程中的面向对象，就是把一组数据结构和处理它们的方法组合成一个整体。当我们理解每一个变量的时候，自然就能想到它有哪些数据结构和操作方法，最简单的类比就像一个人，具有通常的特征以及通常的做事情的方式。Python 中的任何数值、字符串、数据结构、函数、类、模块等都是对象。

2. 一些说明

(1) #符号表示的是注释。

(2) 字符串我们用单引号(' ')或者双引号(" ")都是可以的，三引号(" " " " " ")一般用于放置文档说明(docstring)或多行字符串。比如，我们打开 Spyder 的界面，如图 7-1 所示。

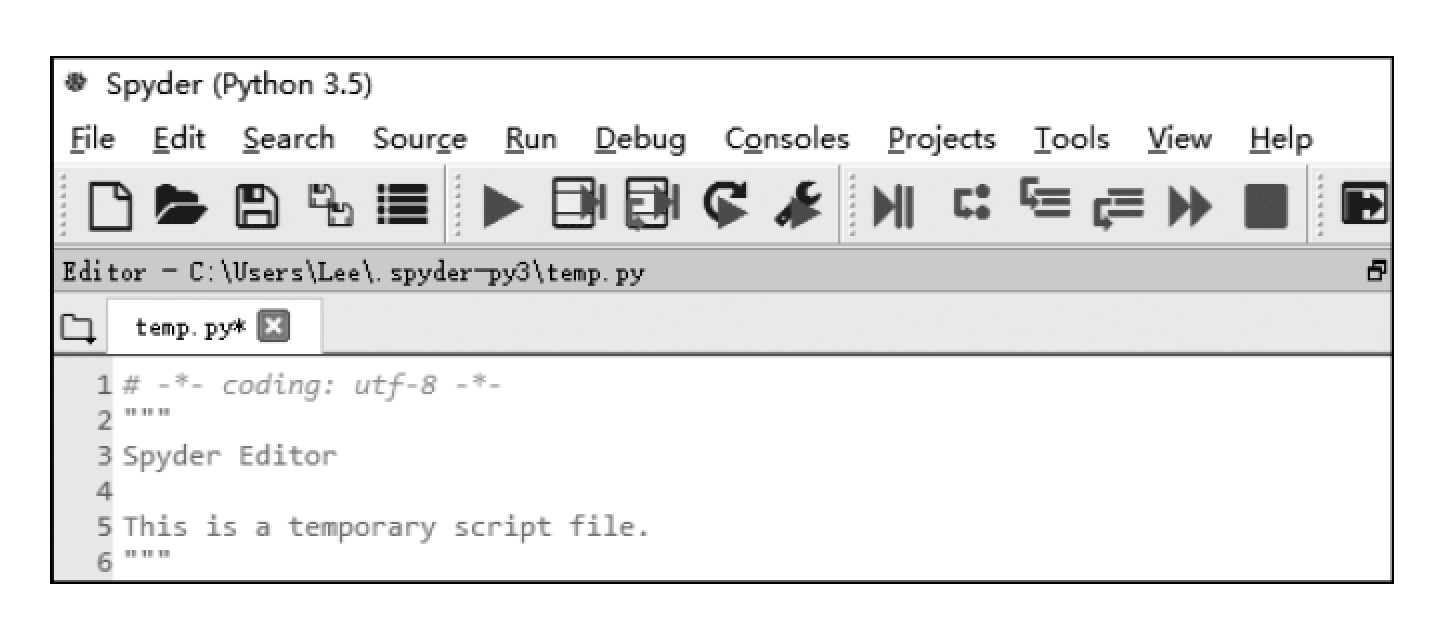


图 7-1

(3) 接下来我们演示全部都用 Jupyter Notebook，这样的安排是方便演示。读者可以直接打开 Jupyter Notebook。

(4) 我们的演示强调的是方法论，探索怎么去操作完成项目，没有很规范地解释每一个命令或函数的操作。我们的目的就是建立一个框架、一个模型，通过这样的框架或者模型快速掌握这项技能。若要让自己更加精通，那是需要不断熟练的。

7.2 本地文件的读取

常用的文件格式包括 txt、xls/xlsx、json、xml、HDF 以及其他可以转换成以上格式的数据文件。

1. 读取 txt 文件

读取文件很简单，但是我们要思路清晰。一般就是打开文件、查看数据、关闭文件，重

点就是记得要关闭文件。这样我们就很容易想到代码怎么写。

（1）打开文件。sh＝open('G:\pybook\index.txt')，如图 7-2 所示。

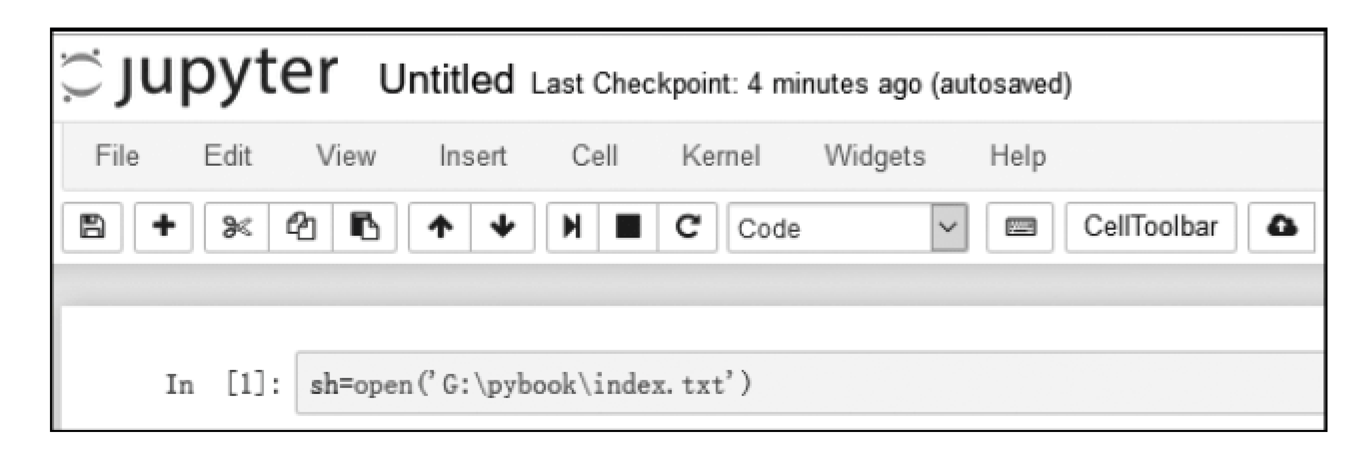


图 7-2

这里 open 就是大家容易想到的“打开”，文件需要指明在哪个位置，所以需要通过一个字符串，说明文件的位置。

如果读者输入“sh”，单击运行，就会发现界面如图 7-3 所示。

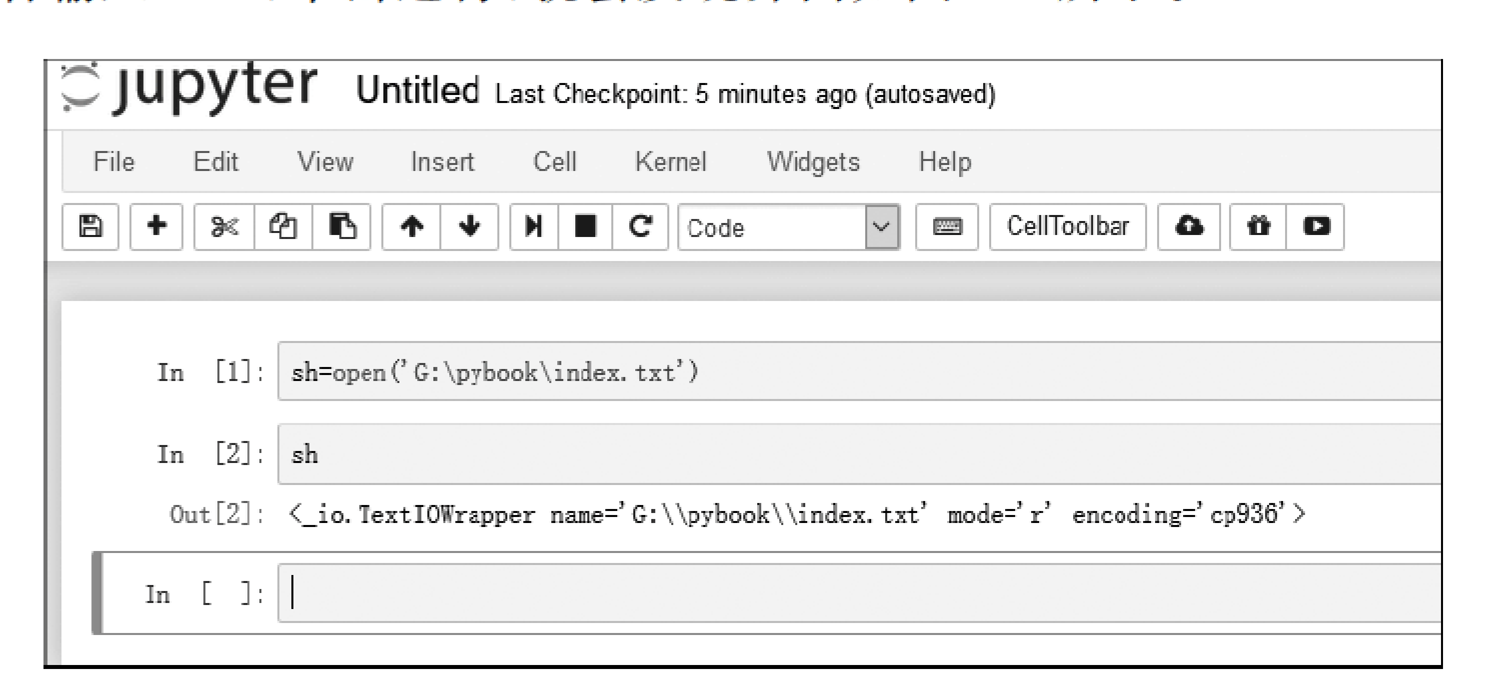


图 7-3

也就是说，sh 是一个对象，我们应该很具体地说要打开对象的哪个属性或者操作方法。那么 sh 到底有哪些属性和操作方法呢？最直接的办法是：输入“sh”，然后输入符号“.”，按下键盘上的“tab”键，如图 7-4 所示。

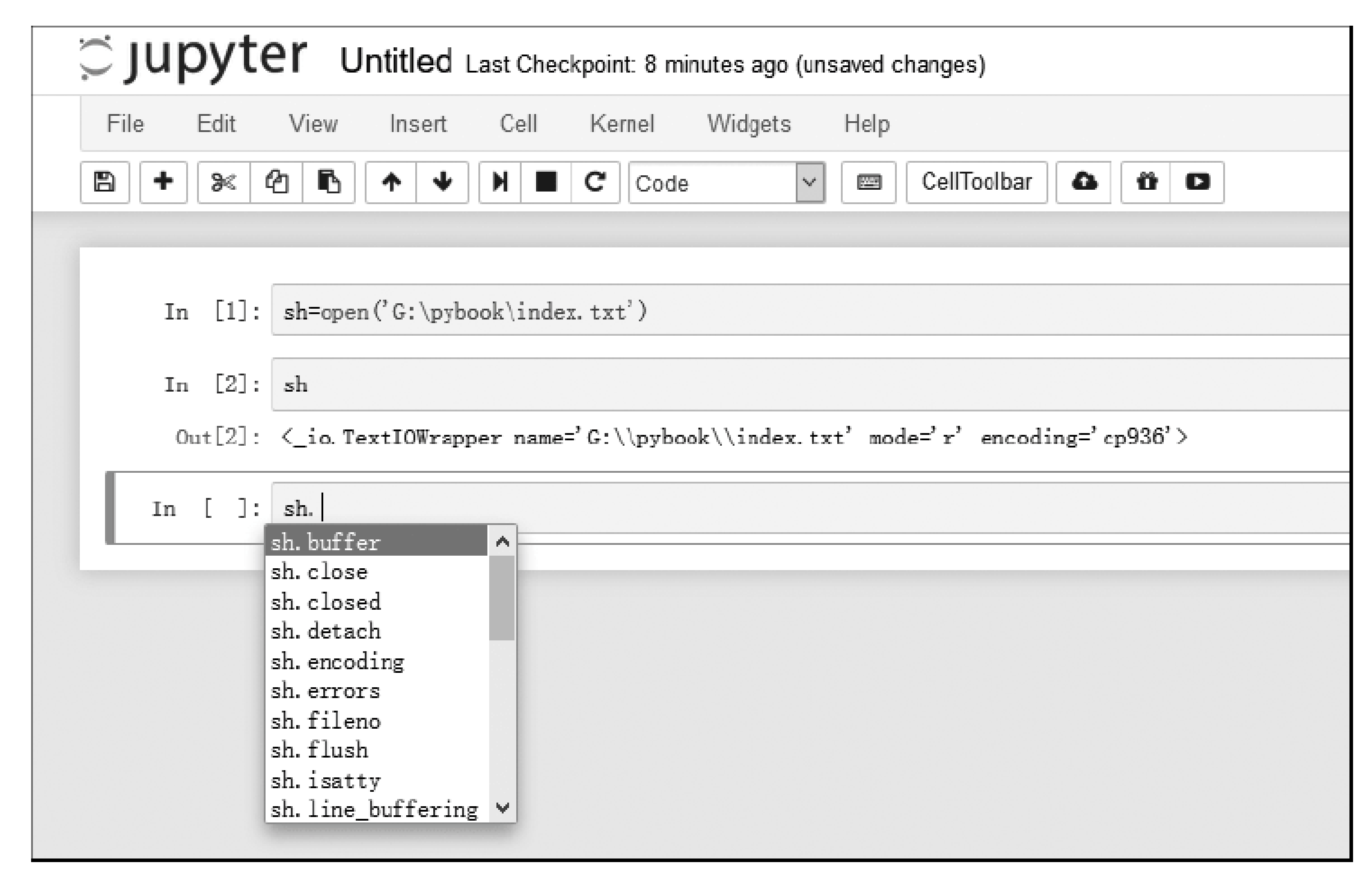


图 7-4

(2) 查看数据。比如我们选择 sh. read(),显示结果如图 7-5 所示。

```
In  [3]: sh.read()

Out[3]: 'num,code,name,change,open,preclose,close,high,low,volume,amount\n0,000001,上证指数,-0.96,3271.8665,3268.9354,3237.4471,3274.1903,3232.2806,
200583223,2621.8405\n1,000002,A股指数,-0.97,3426.087,3423.0096,3389.9542,3428.5144,3384.5289,200163463,2618.6761\n2,000003,B股指数,-0.27,3
50.2097,350.135,349.182,350.6325,348.5784,419759,3.1644\n3,000008,综合指数,-0.87,2813.4248,2811.7415,2787.3802,2816.2685,2780.5465,38839364,
426.3741\n4,000009,上证380,-0.79,5797.7085,5794.5599,5748.6748,5807.4479,5745.1346,43212713,522.1926\n5,000010,上证180,-0.99,7565.3842,7559.
8586,7484.7359,7571.2698,7470.2352,70446381,712.5753\n6,000011,基金指数,-0.4,5849.2142,5847.8353,5824.4571,5852.3577,5821.784,16632663,341.5
28\n7,000012,国债指数,-0.06,160.0978,160.0829,159.993,160.1003,159.9832,73498,0.7392\n8,000016,上证50,-0.94,2371.6054,2369.194,2346.9602,237
3.5439,2342.0186,30767623,331.3074\n9,000017,新综指,-0.97,2762.3167,2759.8351,2733.1681,2764.2698,2728.7886,200134695,2617.8016\n10,000300,
沪深300,-1.03,3485.3114,3481.5066,3445.8051,3488.4969,3441.4685,103755560,1223.6892\n11,000905,中证500,-0.92,6543.5573,6543.3407,6483.2463,6
555.5786,6480.2914,73063706,899.1115\n12,399001,深证成指,-1.03,10632.045,10624.421,10515.414,10650.455,10514.674,20374886009,3381.9337\n13,3
99002,深成指R,-1.03,12560.383,12551.376,12422.599,12582.132,12421.724,8788418520,1299.4677\n14,399003,成份B指,-0.22,6172.19,6170.675,6157.0
43,6182.356,6119.876,12059120,1.1148\n15,399004,深证100R,-1.23,4741.79,4736.744,4678.388,4747.046,4678.211,3333434048,529.4612\n16,399005,中
小板指,-1.17,6842.984,6839.936,6759.796,6855.162,6759.796,1848937338,317.0289\n17,399006,创业板指,-0.87,1968.153,1966.727,1949.661,1981.69,1
949.351,1177127203,254.4567\n18,399008,中小300,-0.93,1481.601,1480.639,1466.847,1484.805,1466.847,4475251777,682.8057\n19,399100,新 指 数,-0
.84,8929.204,8923.028,8847.649,8951.051,8846.403,19933766989,3336.4292\n20,399101,中小板综,-0.72,11878.881,11874.507,11788.991,11910.732,117
88.991,8665275814,1453.8455\n21,399106,深证综指,-0.81,2047.152,2046.307,2029.729,2054.023,2029.591,20374886009,3381.9337\n22,399107,深证A指
,-0.81,2141.667,2140.784,2123.399,2148.876,2123.264,20350459505,3380.0904\n23,399108,深证B指,-0.29,1154.022,1153.403,1150.013,1155.724,1147
.714,24426504,1.8433\n24,399333,中小板R,-1.17,7474.925,7471.596,7384.056,7488.229,7384.056,1848937338,317.0289\n25,399606,创业板R,-0.87,2036
.643,2035.167,2017.507,2050.651,2017.186,1177127203,254.4567\n'
```

图　7-5

这里显示的数据很多,我们会发现,其实每一行的数据都是以“\n”结束的。最后,我们需要关闭文件,因为如果你不关闭文件,其他操作方法就是不可行的。举个例子,假设你没有关闭文件,然后你在本地删除文件时,就会弹出对话框,如图 7-6 所示。

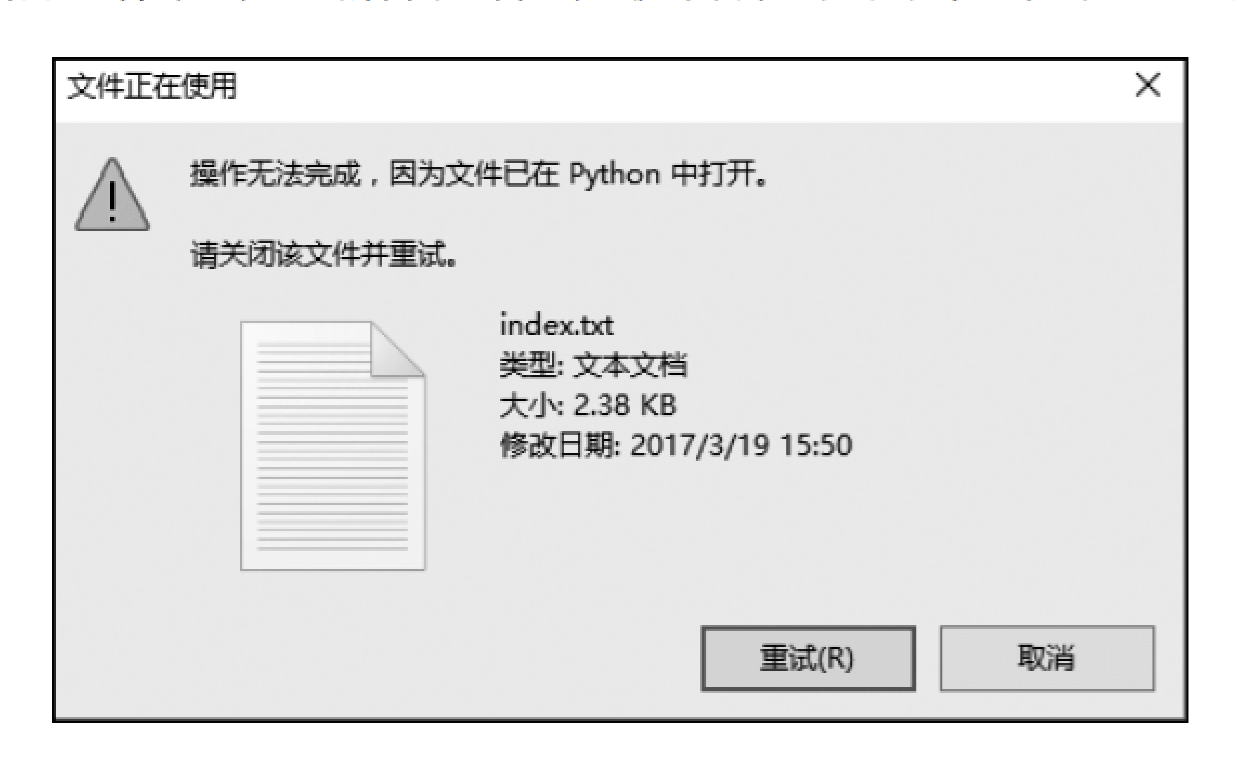


图　7-6

(3) 关闭文件。使用 sh. close(),如图 7-7 所示。

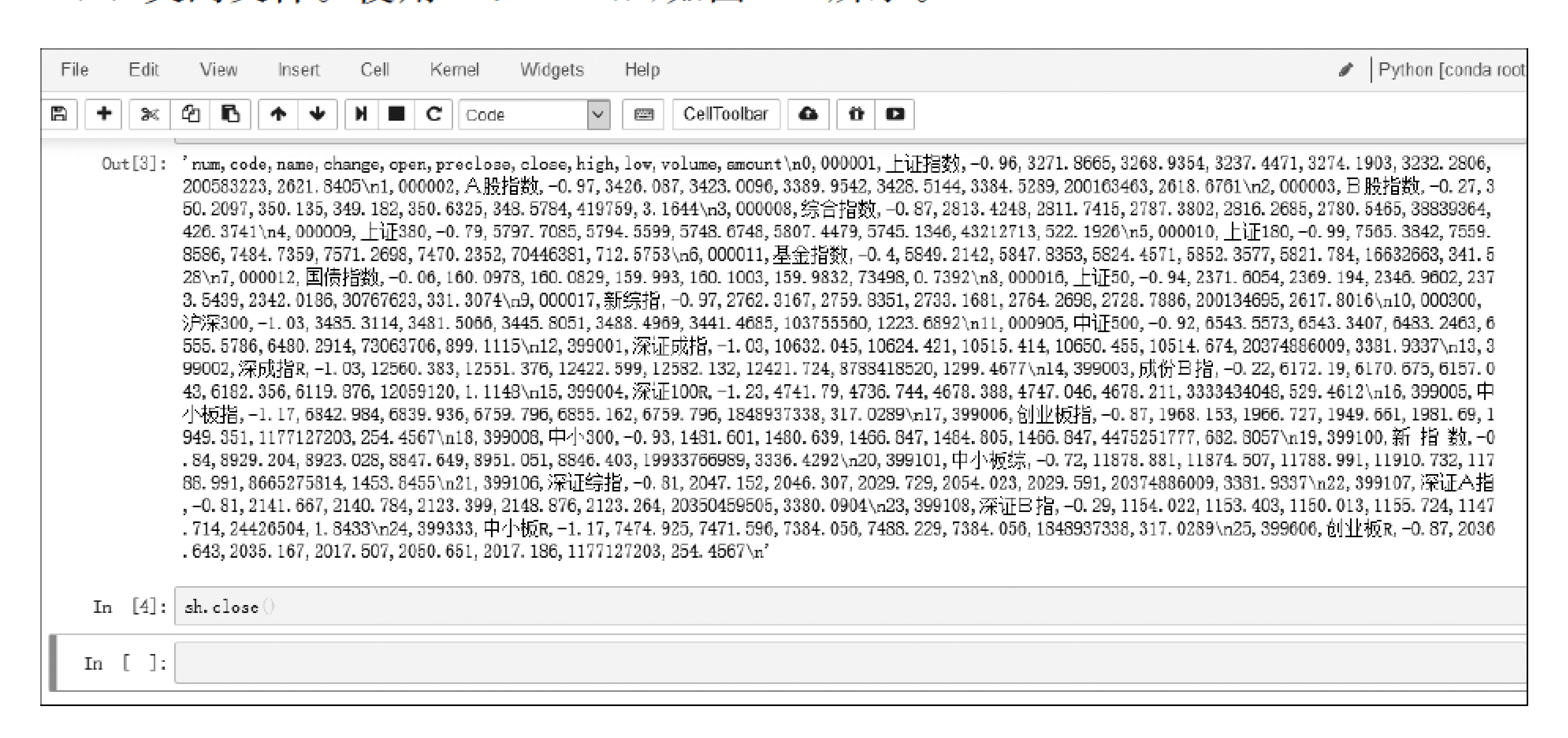


图　7-7

2. 读取 CSV 文件

CSV(comma separated values)以纯文本形式存储的表格数据(以逗号作为分隔符),通常第一行为列名。

(1) 打开文件:fp=open("G:\pybook\index.csv")。

(2) 查看数据:fp.read(),如图 7-8 所示。

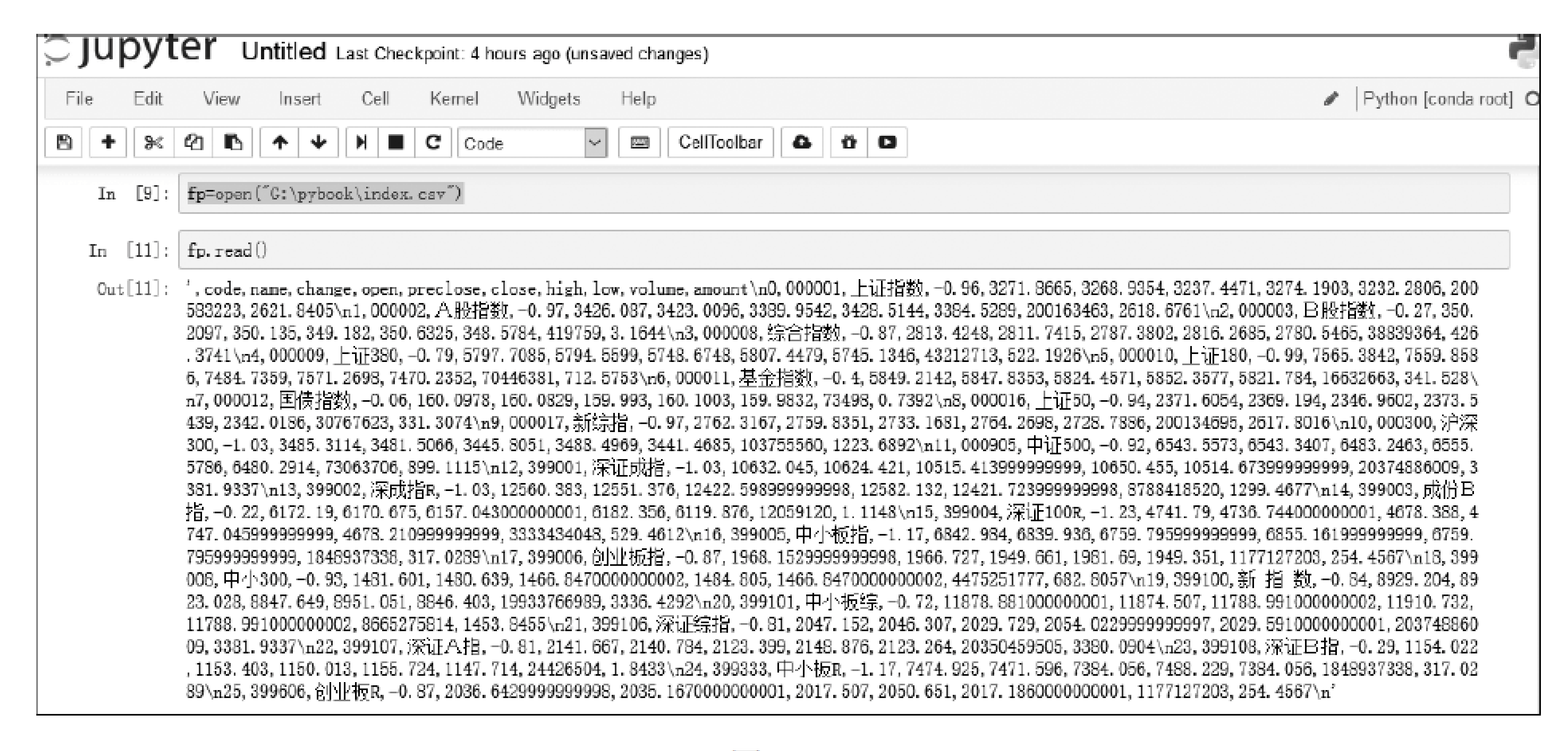


图 7-8

(3) 关闭文件:fp.close()。

其实,读者发现读取 txt 文件和 csv 文件基本上是一样的,我们可以这么看,凡是本地文件,我们都可以把 txt 文件和 csv 文件看成一样,用相同的操作方法。

3. 读取 Excel 的数据

如果我们仍然按照上面这个步骤,会遇到一些问题,如图 7-9 所示。

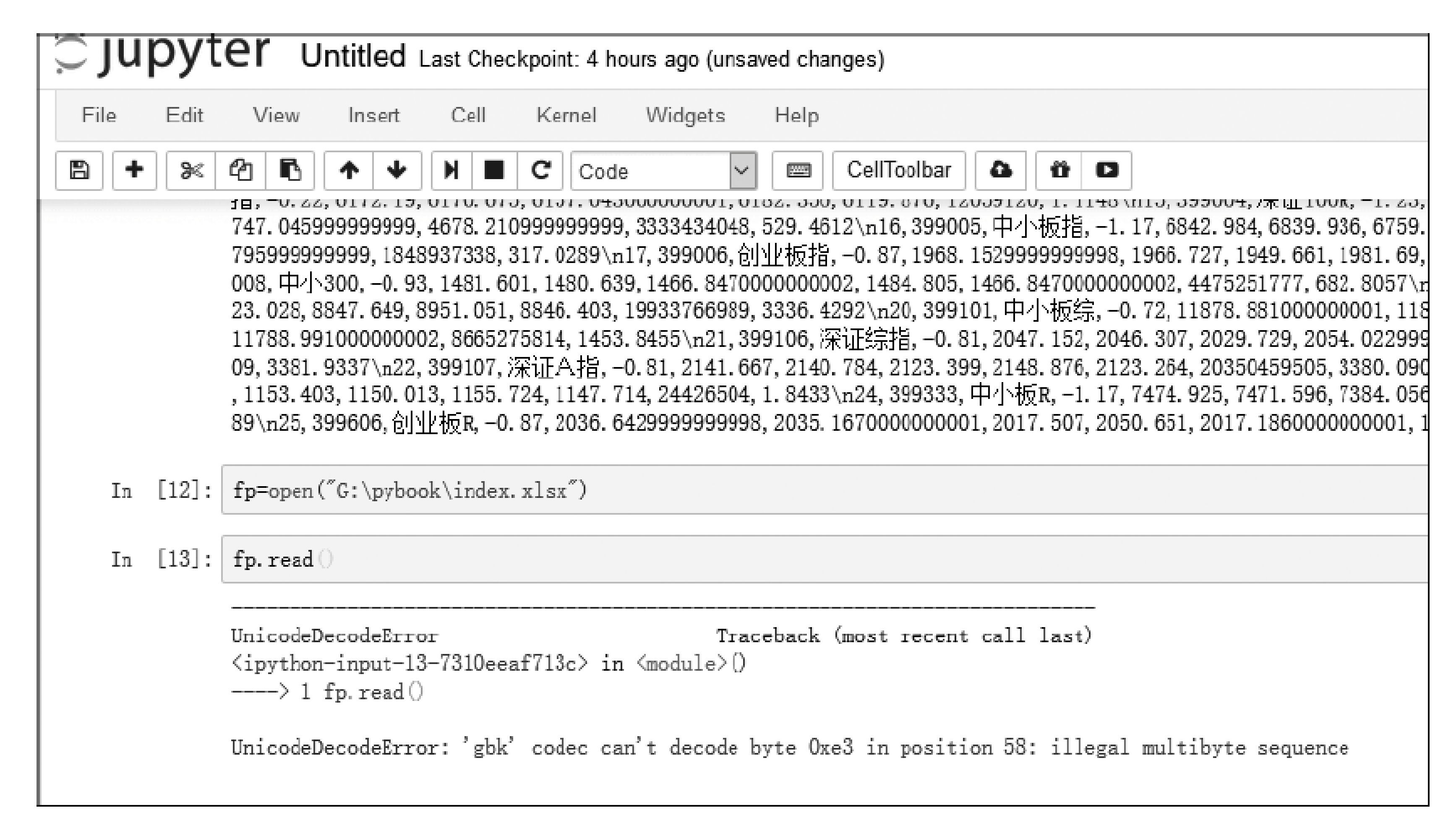


图 7-9

但是，本书就不介绍读取 Excel 格式的文件了，理由是一方面我们可以把 Excel 文件转换成 CSV 文件；另一方面也有读取 Excel 文件的一些第三方包，而且 Excel 太庞大了。

4. 读取其他文件

至于读取 json 和 HDF 等文件，我们就不直接介绍了，感兴趣的读者自行百度搜索进行学习，或者遇到需要解决的问题，再自行查资料。

第 8 章 数据类型和数据结构

8.1 基本数据类型

业内有一句话，一般的程序员考虑的是代码，高手关注的是数据结构以及它们的关系。本章介绍 Python 的基本数据类型和数据结构。Python 解释程序本身自带丰富的数据结构，NumPy 和其他库又增添了许多有价值的数据结构。

Python 是一种动态型语言，也就是说，Python 解释程序在程序运行的时候推知对象的类型。静态类型的语言是必须在编译前就清楚对象类型。

1. 整数

Python 最基本的数据类型毫无疑问就是整数了，如图 8-1 所示。

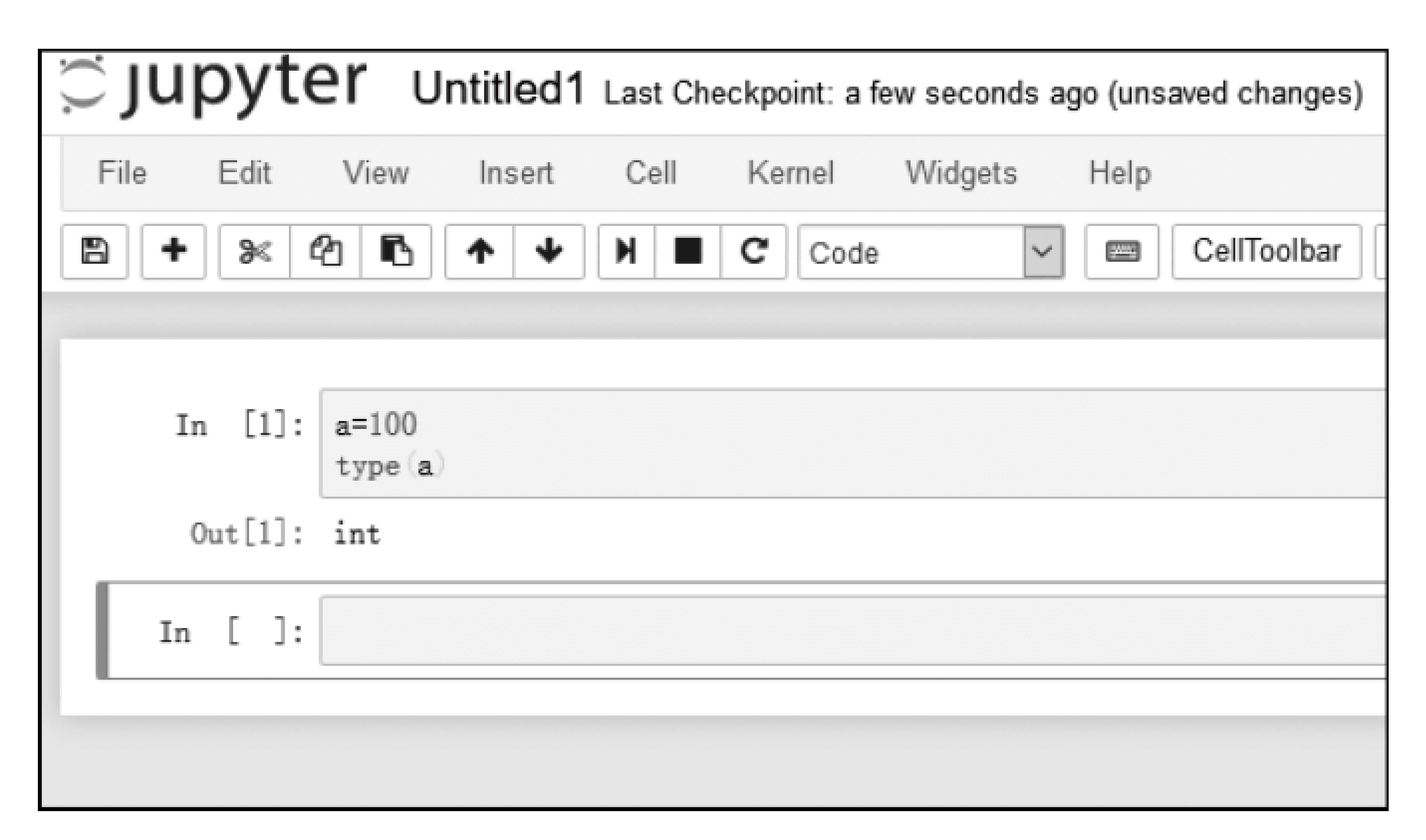


图 8-1

函数 type 是系统提供的，我们把变量输入给 type，然后 type 就告诉我们该变量的数据类型。

记住一句话"Python 中，一切都是对象"。对于刚定义的简单对象也是有操作方法的。

一般来讲一个对象可能有很多方法，我们完全没必要去记住这些方法，一般就是前面说的，输入一个点"."，然后按 tab 键，就可以看到显示的所有方法了，如图 8-2 所示。

Python 的特点之一就是整数可以任意大，Python 处理这些数字完全不是问题，当然从技术上讲，这是一个很大的对象。例如，考虑围棋的各种可能性为 2^361。

注意：Python 中，我们用的是 2＊＊361，而不是 2^361，如图 8-3 所示。

还需要注意的是，int 的运算结果返回不一定是 int 对象。这个有时候可能不是那么容易被发现或者难以检查到错误，如图 8-4 所示。

然而，如图 8-5 这种情况就是读者需要注意的。

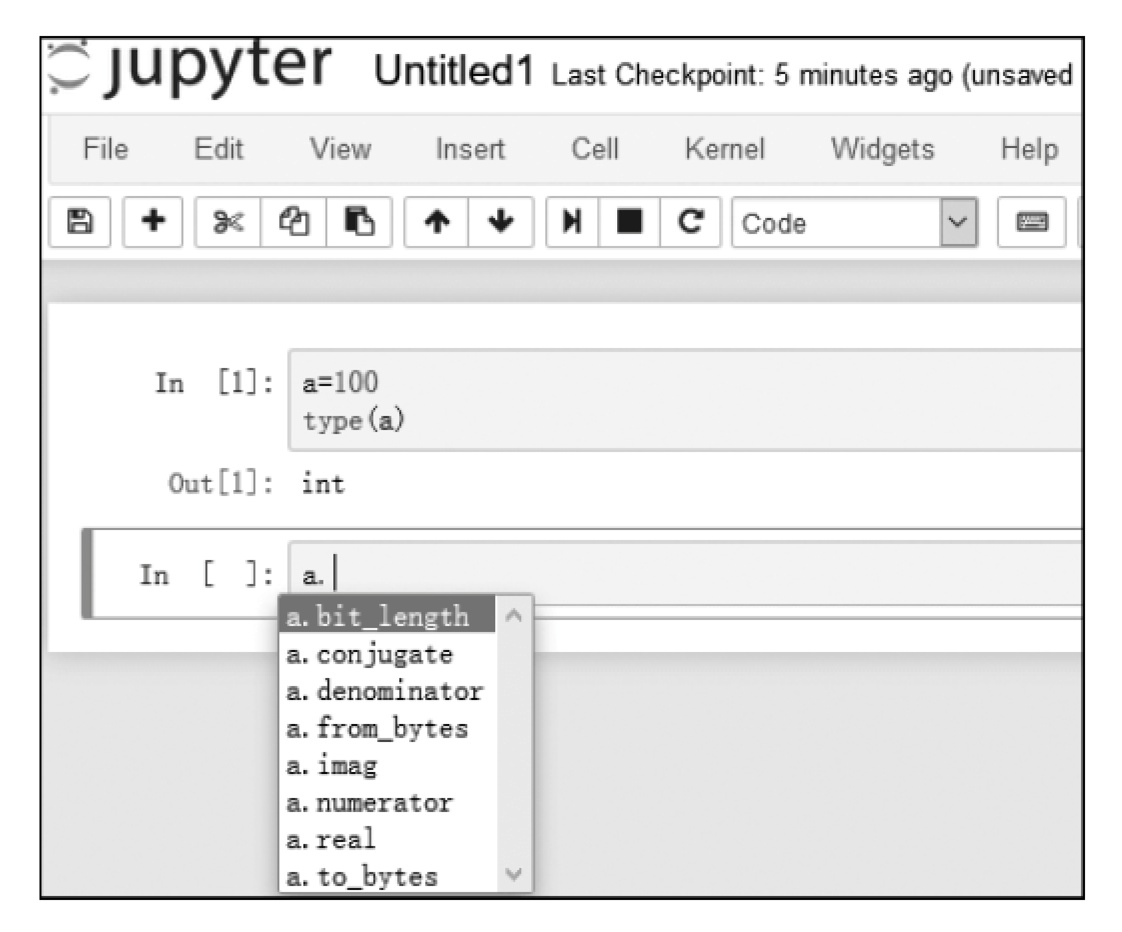


图 8-2

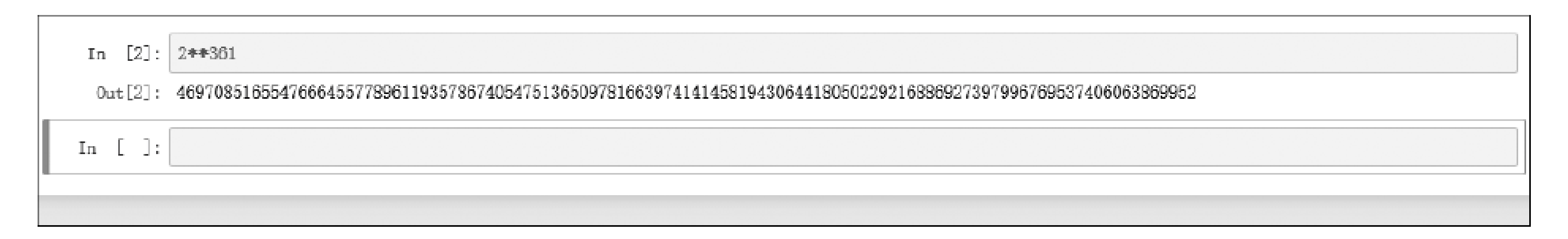


图 8-3

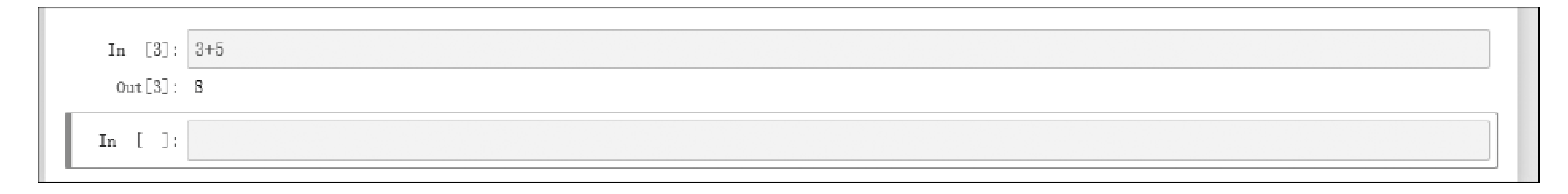


图 8-4

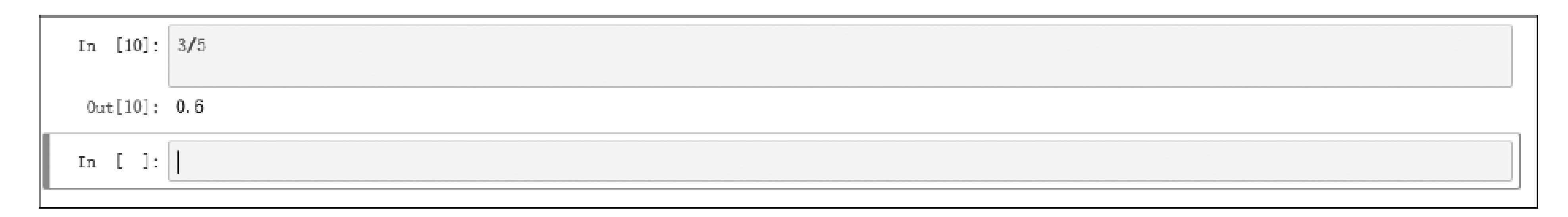


图 8-5

如果需要得到的结果还是 int 型，那么需要使用“//”，如图 8-6 所示。

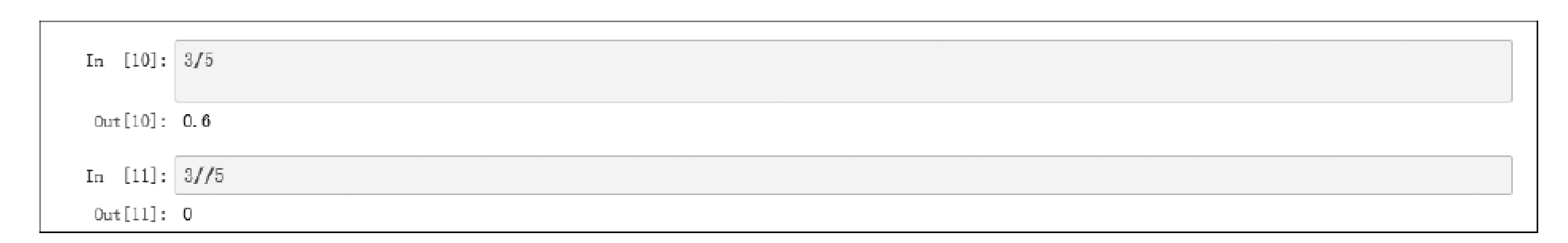


图 8-6

2. 浮点型

浮点型是我们最常用到的类型。

(1) 如果我们需要使用浮点型,就可以在整数后面输入“.”,如图 8-7 所示。

```
In  [12]: b=100.
          type(b)
Out[12]: float
```

图 8-7

当然也可以直接使用 float 函数,如图 8-8 所示。

```
In  [12]: b=100.
          type(b)
Out[12]: float

In  [13]: x=float(100)
          type(x)
Out[13]: float
```

图 8-8

(2) 既然是浮点数,也就涉及精确度的问题。举个例子,如图 8-9 所示。

```
In  [16]: a=0.37
          b=0.1
          a+b
Out[16]: 0.47

In  [17]: a=0.35
          b=0.1
          a+b
Out[17]: 0.44999999999999996
```

图 8-9

读者可能觉得很奇怪,有一些浮点数相加符合我们的预期结果,有一些就不对。

其实,出现上述结果的原因是浮点数在计算机内部是通过二进制形式表示的,也就是说一个近似表示,这就会导致结果不精确。

由于这个问题在金融行业非常重要,有时候必须保证数值的精确(至少尽可能达到最佳)。这个在加总一组数量很多的数值时就显得很重要了。在这种情况下,某个种类或者精度的表示误差可能汇集起来,造成和原始值存在一个显著误差。

肯定也有解决此问题的方法。下面我们需要了解 Decimal 模块,该模块为浮点数提供了任意精度的对象,以及使用这些数值时处理精度问题的多个选项。

第一步:导入 Decimal 模块,如图 8-10 所示。

```
import decimal
from decimal import Decimal
```

```
In  [18]: import decimal
          from decimal import Decimal

In  [ ]:
```

图 8-10

第二步:了解默认的精度位数,如图 8-11 所示。

```
In  [19]: decimal.getcontext()
Out[19]: Context(prec=28, rounding=ROUND_HALF_EVEN, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[], traps=[InvalidOperation, DivisionByZero
         , Overflow])

In  [ ]:
```

图 8-11

这里我们只关注 28 这个数字，就是说默认的精度是 28 位。举个例子，如图 8-12 所示。

```
In  [22]: d=Decimal(1)/Decimal(9)

In  [23]: d

Out[23]: Decimal('0.1111111111111111111111111111')

In  [ ]:
```

图 8-12

而如果我们不使用这个模块的话，也就是说，一般情况下，Python 运行的平台精度标准是 15 位的精度，如图 8-13 所示。

```
In  [24]: 1/9
Out[24]: 0.1111111111111111

In  [ ]:
```

图 8-13

第三步：确定要使用的精度，如图 8-14 所示，结果就是保留小数点后两位数字。

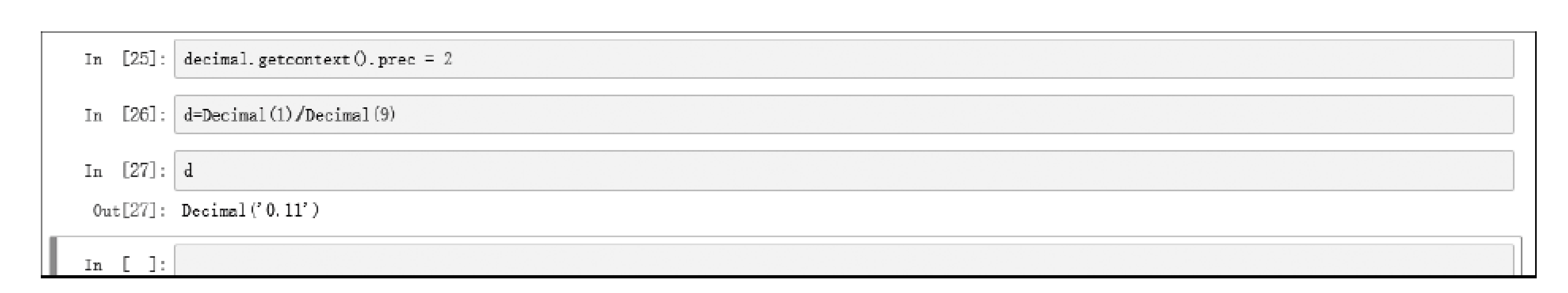


图 8-14

3. 字符串

前面介绍了整数和浮点数，现在我们就可以讲解一下文本了。Python 就是表示文本的，如图 8-15 所示。

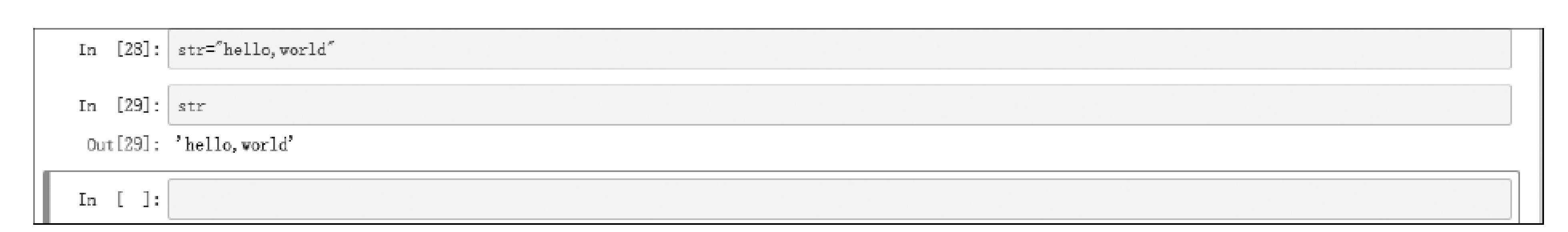


图 8-15

如前面所说的，对于字符串我们可以使用单引号或者双引号。字符串也有很多种表示方法，如图 8-16 所示。

比如第一个函数，表示的是将第一个词转换成大写，如图 8-17 所示。

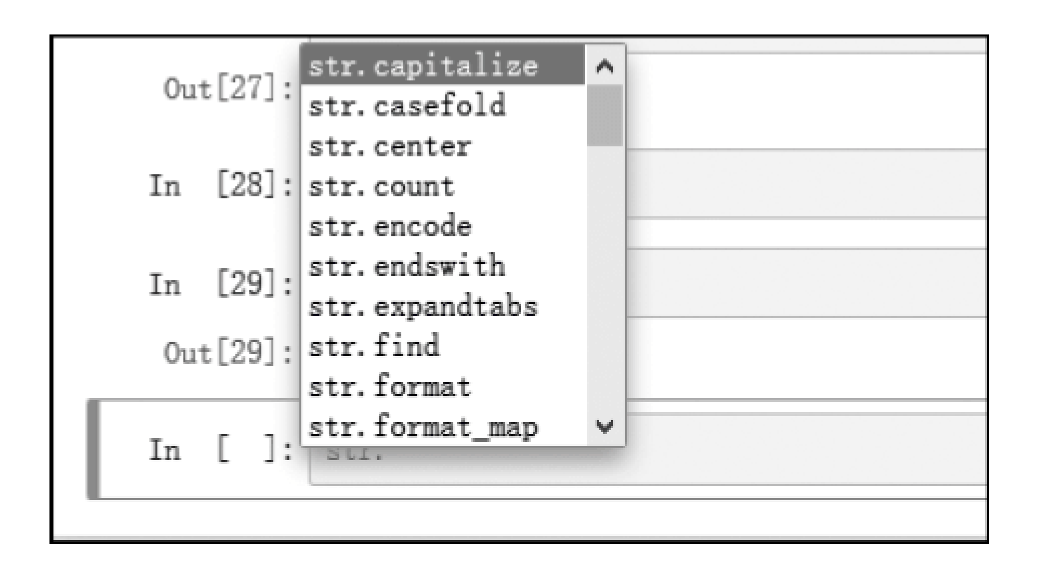


图 8-16

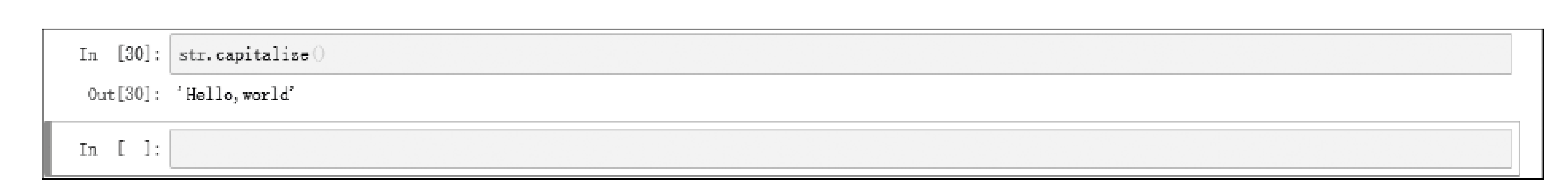


图 8-17

其他的,读者可以自行百度进行学习了解,特别是在解决具体问题的时候。

8.2 基本数据结构

一般的原则是,数据结构是包含其他对象的对象。Python 提供的数据结构包括以下几种:

元组(tuple)。任意对象的集合,只有少数可用的方法。

列表(list)。任意对象的集合,有许多可用的方法。

字典(dict)。键-值存储对象。

集合(set)。其他独特对象的无序集合对象。

1. 元组

元组是一种高级的数据结构,但是其应用还是相当的简单有限,形式是使用()定义对象,如图 8-18 所示。

```
In [31]: t=(1,3.4,"hello")
         type(t)
Out[31]: tuple
```

图 8-18

甚至,我们还可以不需要(),直接使用逗号“,”,如图 8-19 所示。

```
In [31]: t=(1,3.4,"hello")
         type(t)
Out[31]: tuple

In [32]: t=1,3.4,"hello"
         type(t)
Out[32]: tuple
```

图 8-19

和 Python 中几乎所有的数据结构相似，元组有内建的索引，利用索引可以读取元组的单个元素或者多个元素。这里需要记住的是 Python 使用的是从 0 开始编号的，这一点和 MATLAB 不一样，MATLAB 是从 1 开始的。比如上面的第三个元素在索引位置 2 上面，如图 8-20 所示。

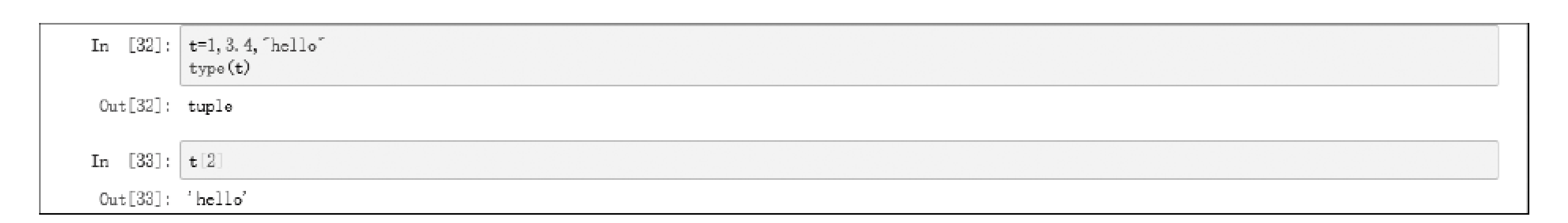


图 8-20

我们看到元组的方法很少，如图 8-21 所示。



图 8-21

可能读者会感到很奇怪，为什么需要元组这个对象呢？这里我们先了解列表后再一起做个比较。

2. 列表

与元组相比，列表类型的对象更灵活、更强大。从金融的角度看，许多工作只能用列表对象完成，比如存储股票的开高低收数据。列表对象通过方括号定义，基本功能和行为与元组类似，如图 8-22 所示。

```
In [34]: t=[1,3.4,"hello"]
         type(t)
Out[34]: list

In [35]: t[2]
Out[35]: 'hello'

In [ ]: |
```

图 8-22

可以看到，列表的方法有多种，如图 8-23 所示。

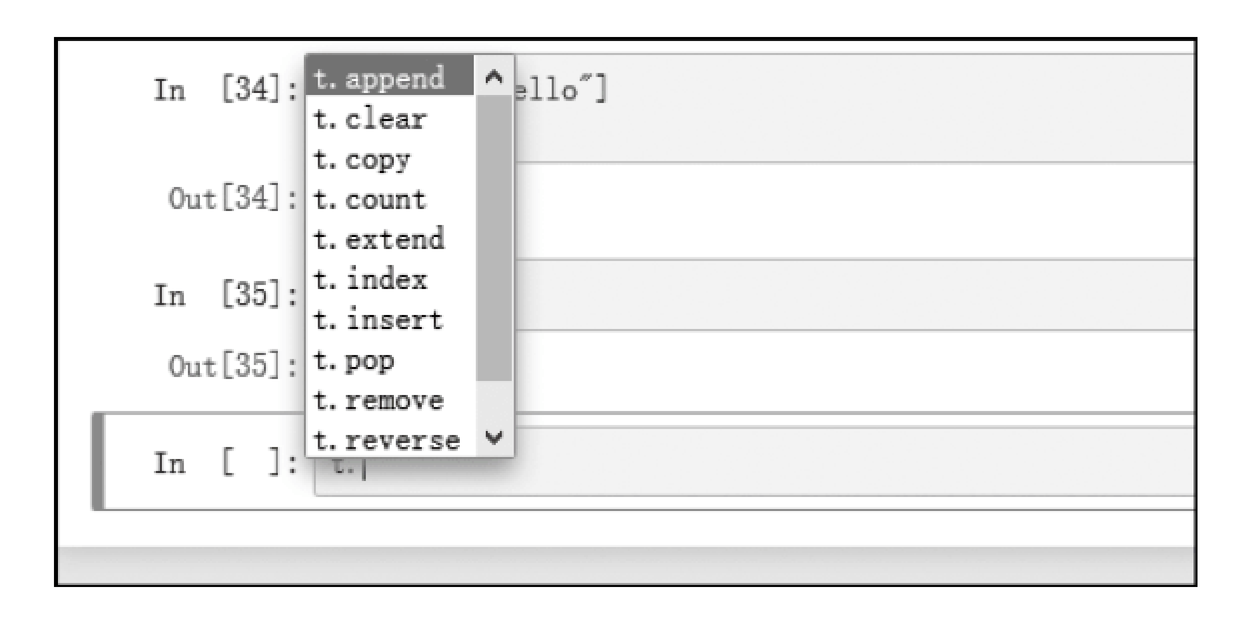


图 8-23

比如添加数据，如图 8-24 所示。

```
In [36]: t.append([3,4,5])

In [37]: t
Out[37]: [1, 3.4, 'hello', [3, 4, 5]]
```

图 8-24

比如插入数据,如图 8-25 所示。

```
In [38]: t.insert(1,"Insert")
         t
Out[38]: [1, 'Insert', 3.4, 'hello', [3, 4, 5]]
```

图 8-25

上面演示的是在第 1 个索引的前面插入"insert",移除数据如图 8-26 所示。

```
In [39]: t.remove("hello")
         t
Out[39]: [1, 'Insert', 3.4, [3, 4, 5]]
```

图 8-26

比如切分数据或者说切片,这里我们更愿意说切分数据。切分数据的意思是把数据集分解为几个部分,这个在金融数据分析中会经常用到,如图 8-27 所示。

```
In [42]: t[1:3]
Out[42]: ['Insert', 3.4]
```

图 8-27

这里需要注意的是方括号的定义,它是一个左闭右开的区间,也就是说从左边索引开始,到右边索引结束,但是不包括右索引的对象。如上图所示:左边索引的是"insert",右边索引 3 的应该是[3,4,5],但是不包括这个索引的对象,这个需要注意。

总结一下,关于列表的常见操作方法如表 8-1 所示。

表 8-1

方　法	参数	返回或结果
t[i]=x	[i]	用 x 替代第 i 元素
append	(x)	在对象后附加 x
count	(x)	对元素 x 统计出现的次数
insert	(i,x)	在索引 i 之前插入元素 x
remove	(i)	删除索引为 i 的元素
reverse	(x)	所有项目的逆向顺序

读者可能会问元组和列表有什么区别呢,为什么要这么设计呢?这里的差异就是列表是可变的,元组是不可变的。这就类似于一个 Excel 表格的行和列,每列基本上保持不变,变动的更多的是行数的变化。这里我们可以简单地理解为:list 类似于 Excel 表格的某一列数据,元组类似于第一行的列标签或每一行代表的数据。

3. 字典

字典对象就是按照键码读取的数据字典，也是一种可变序列。字典用的是大括号，如图 8-28 所示。

```
In  [60]: dt={"name":"二娃","age":"18","country":"china"}
          dt
Out[60]: {'age': '18', 'country': 'china', 'name': '二娃'}
```

图 8-28

从上面的例子可以看到字典的对象是没有顺序的，如图 8-29 所示。

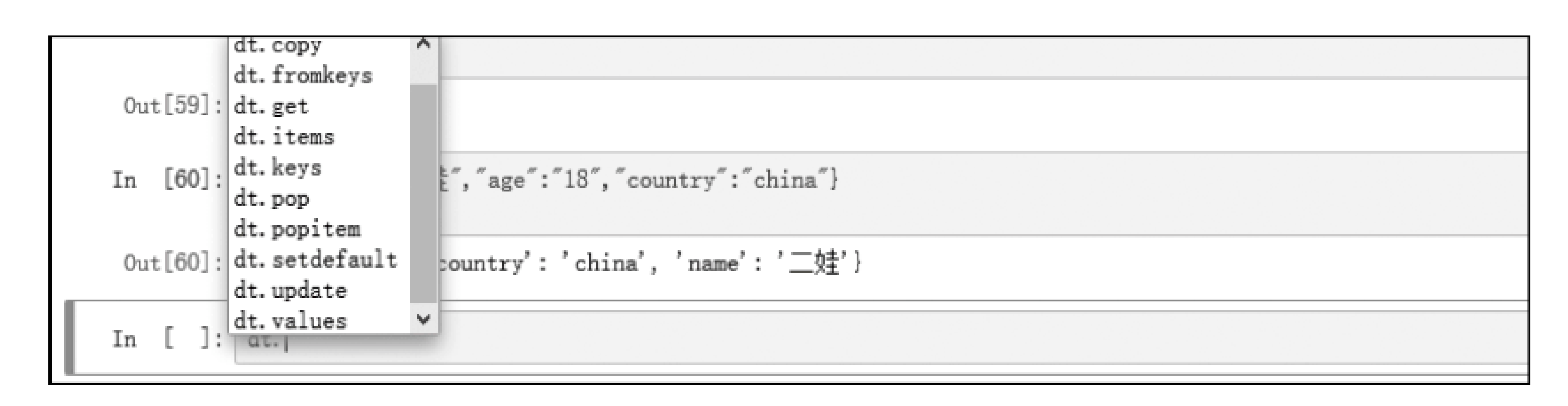


图 8-29

字典的方法有多种，如图 8-30 所示。

```
In  [62]: dt.keys()
Out[62]: dict_keys(['country', 'age', 'name'])

In  [63]: dt.values()
Out[63]: dict_values(['china', '18', '二娃'])

In  [64]: dt["age"]
Out[64]: '18'
```

图 8-30

4. 集合

这种数据结构没有多大用处，或者我们做金融行业的可能用得不多。不过，集合的好处是可以去掉列表对象中的重复元素，如图 8-31 所示。

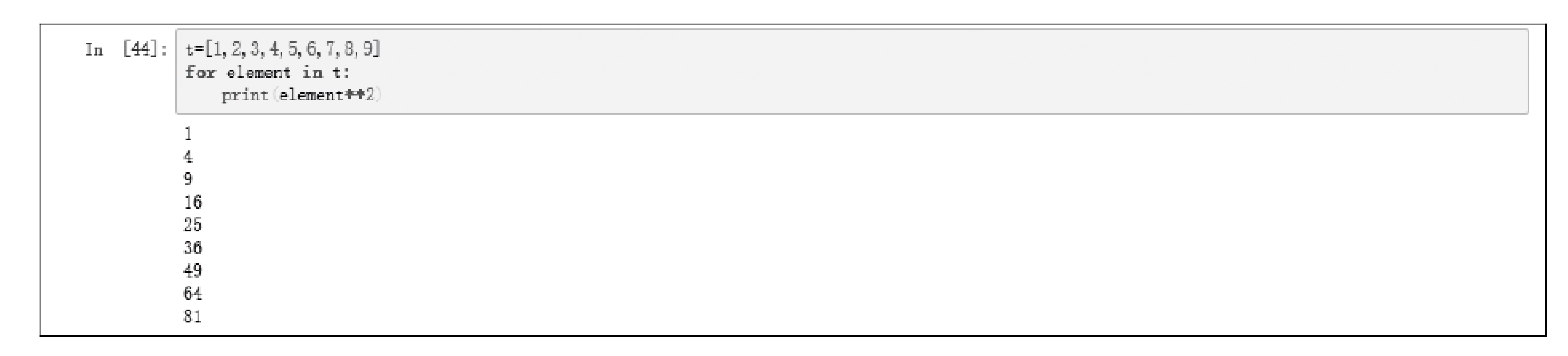
```
In  [65]: ret=[1,2,2,3,3,3,4,4,4,4,5,5,5,5,5]
          s=set(ret)

In  [66]: s
Out[66]: {1, 2, 3, 4, 5}
```

图 8-31

5. 控制结构

这里的控制结构本身(这里只介绍 for 循环)就应该专门讨论，但是在 Python 中，最适合在列表对象基础上进行介绍。这是因为循环一般来说都是在列表对象上进行的，如图 8-32 所示。



```
In [44]: t=[1,2,3,4,5,6,7,8,9]
         for element in t:
             print(element**2)

1
4
9
16
25
36
49
64
81
```

图 8-32

这里的循环具有很高的灵活性，需要注意的是没有大括号等，在 for 循环中第二行有缩进（空白行）。

在 Python 中，可以在任意列表对象上循环，不管这些对象是什么。

6. 定义函数

在 Python 中，定义一个函数要使用 def 语句，依次写出函数名、括号、括号中的参数和冒号。然后，在缩进块中编写函数体，函数的返回值用 return 语句返回。

举个简单的例子，考虑返回输入 x 的平方的函数 f，如图 8-33 所示。

```
In [45]: def f(x):
             return x**2
         f(12)

Out[45]: 144
```

图 8-33

请注意，函数体内部的语句在执行时，一旦执行到 return 时，函数就执行完毕，并将结果返回。因此，函数内部通过条件判断和循环可以实现非常复杂的逻辑。

如果没有 return 语句，函数执行完毕后也会返回结果，只是结果为 None。

return None 可以简写为 return。

7. 过滤、映射和归纳

Python 提供一些用于函数式编程支持的工具，也就是在列表对象应用某个函数。这些工具包括过滤、映射和归纳等。

(1) 熟练掌握 lambda 函数

首先，要明白 lambda 表达在 Python 中是作为一个匿名函数的构造器而存在。其次，要明白 lambda 表达式的常用场景是 lambda 表达式对应函数的使用次数非常有限（因此，没有必要专门定义一个非匿名函数），同时保证了代码的简洁性。编程中提到的 lambda 表达式，通常是在需要一个函数，但是又不想费神去命名一个函数的场合下使用，也就是指匿名函数。在用 Python 中不得不涉及 lambda 函数，我们可以直接地理解为临时的一个函数，先举个例子，如图 8-34 所示。

```
In [46]: ret=lambda x:x**2
         ret(12)

Out[46]: 144
```

图 8-34

从上面的例子中可以看到，lambda 的结构就是自变量，然后冒号，后面提供关于自变量的一个函数关系，上面的例子显示的是关于 x 的平方关系。

lambda 匿名函数经常被用到 filter()、map()、reduce()、sorted()函数中，这些函数的共同点是均需要函数型的参数，lambda 表达式正好适用。举个例子，如图 8-35 所示。

```
In  [49]: ret=map(lambda x:x**2, range(10))
          ret
Out[49]:  <map at 0x20433752940>

In  [50]: ret=map(lambda x:x**2, range(10))
          list(ret)
Out[50]:  [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

图 8-35

上面的例子中，对 lambda 函数应用于 range(10)，这里需要注意的就是前面说的 Python 中一切皆对象，返回的结果就是一个对象，如果我们直接看不到结果，但是我们可以通过转换成列表进行查看。

(2) 重要的 range()函数

比如我们需要生成 1～100 的一个自然数列，那我们最好利用 range 函数，如图 8-36 所示。

```
In  [53]: range(1,5)
Out[53]:  range(1, 5)

In  [54]: list(range(1,5))
Out[54]:  [1, 2, 3, 4]
```

图 8-36

从图 8-36 可以看到，range 函数返回的也是一个对象，如果需要看结果，我们需要转换成 list 对象。在 Python 中，大家最熟悉的可能就是 list 对象了。

另外，大家看到的也是括号里面显示的是一个左闭右开的数列(1,5)显示的是 1,2,3,4，不包括 5。

(3) filter()函数

如图 8-37 所示。

```
In  [55]: ret=filter(lambda x:x%2==0, range(1,10))
          list(ret)
Out[55]:  [2, 4, 6, 8]
```

图 8-37

filter 函数的作用是从列表对象中筛选出符合条件的对象，比如上述例子就是从 1,2,3,4,5,6,7,8,9 中选出偶数，结果为 2,4,6,8。

(4) map()函数

如图 8-38 所示。

map()函数的作用是对 list 对象进行对错的判断，符合条件的返回 True，错误的返回 False。

```
In [56]: ret=map(lambda x:x%2==0,range(1,10))
         list(ret)
Out[56]: [False, True, False, True, False, True, False, True, False]
```

图 8-38

(5) reduce()函数

如图 8-39 所示。

```
In [59]: from functools import reduce
         ret=reduce(lambda x,y:x+y,range(1,10))
         ret
Out[59]: 45
```

图 8-39

这里特别需要注意的是关于 reduce()函数需要引入 functools 模块。如 from functools import reduce。

同时,reduce()函数的作用是针对列表对象的所有元素应用一个函数,返回一个值,这里就不需要 list()函数[①]了,如图 8-40 所示。

```
In [65]: ret=[1,2,2,3,3,3,4,4,4,4,5,5,5,5,5]
         s=set(ret)

In [66]: s
Out[66]: {1, 2, 3, 4, 5}
```

图 8-40

① list()是 Python 里面的数据结构和列表函数。

第 9 章

NumPy 基础

NumPy 是 Python 的关于高性能科学计算和数据分析的基础包。最重要的是，提供了矩阵运算的功能。其实，list 已经提供了类似于矩阵的表示形式，不过 NumPy 为我们提供了更多的函数。我们主要是介绍 NumPy 的数组和矩阵计算。

9.1 NumPy 数组

9.1.1 数组对象

先举个例子，如图 9-1 所示。

```
In [69]: import numpy as np
         ret=np.arange(10)
         ret
Out[69]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

图 9-1

这里需要注意的有以下几点：

(1) 导入 NumPy。我们通过这样的形式进行导入：import numpy as np。这里就相当于把 numPy 简写成 np。

(2) 这里返回的结果是 array，读者可能注意到这里用了 np.arange()。

range()函数和 arange()函数有什么区别呢？

记住一个最简单的区别。range()不支持步长为小数，np.arange()支持步长为小数，如图 9-2 所示。

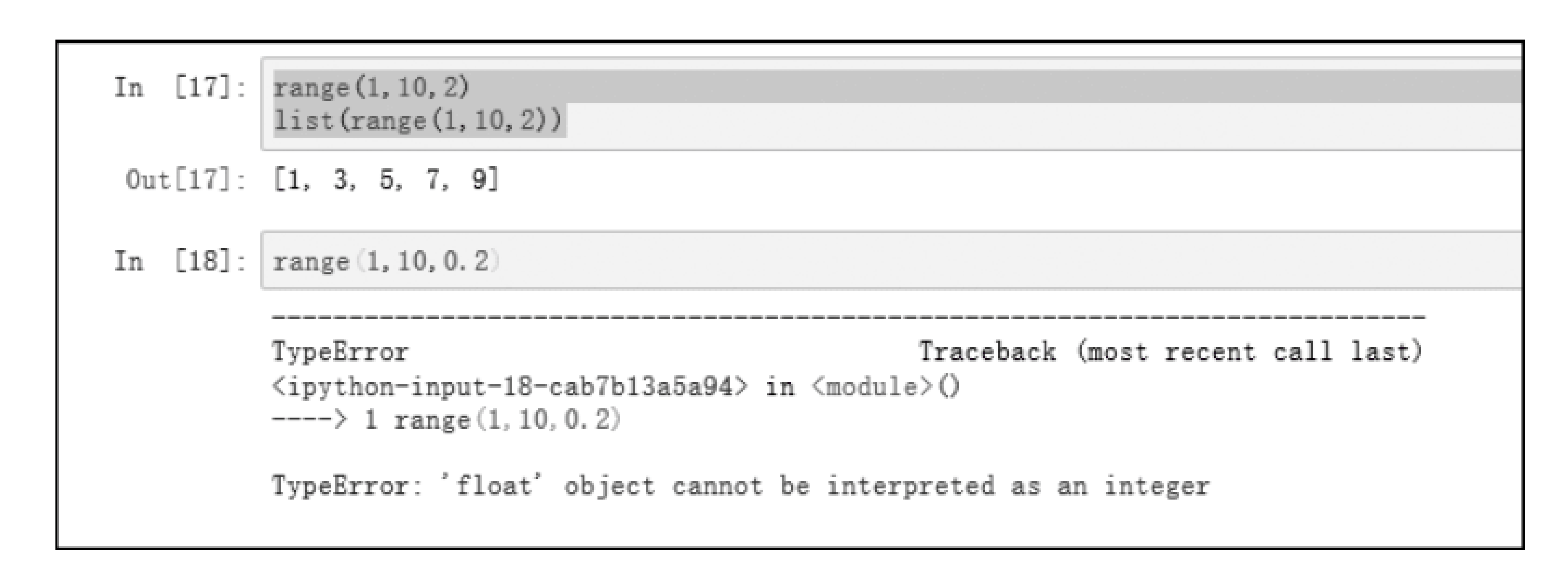


图 9-2

我们再看 NumPy 中的 arange 函数，如图 9-3 所示。

Python 中的 list 是 Python 的内置数据类型，NumPy 的数组和 list 之间有什么区别呢？

```
In  [19]: import numpy as np
          np.arange(1,10,0.2)

Out[19]: array([ 1. ,  1.2,  1.4,  1.6,  1.8,  2. ,  2.2,  2.4,  2.6,  2.8,  3. ,
                 3.2,  3.4,  3.6,  3.8,  4. ,  4.2,  4.4,  4.6,  4.8,  5. ,  5.2,
                 5.4,  5.6,  5.8,  6. ,  6.2,  6.4,  6.6,  6.8,  7. ,  7.2,  7.4,
                 7.6,  7.8,  8. ,  8.2,  8.4,  8.6,  8.8,  9. ,  9.2,  9.4,  9.6,
                 9.8])
```

图 9-3

简单地说就是 list 中的数据类型不必相同，可以是任何对象；而 array 中的数据类型必须全部相同。这里我们最好这么去理解，array 其实是更适合去解决矩阵计算而设置的，比如我们知道的矩阵里面每个元素都必须是相同的类型，而 np.arange()表示的就是一个矩阵，我们可以看作多维数组，也可以看作矩阵，当然矩阵其实是多维数组的一个子类。看作是矩阵，很多操作方法可能会比较好理解。

9.1.2 创建多维数组

创建一个简单的多维数组。

这里可以看到，我们创建了一个多维数组。Shape 属性显示的可以看成是一个 2 行 4 列的矩阵，如图 9-4 所示。

```
In  [72]: data=[[1,2,3,4],[5,6,7,8]]
          arr=np.array(data)
          arr

Out[72]: array([[1, 2, 3, 4],
                [5, 6, 7, 8]])

In  [73]: arr.shape

Out[73]: (2, 4)
```

图 9-4

大家可以看到 arr 的方法是很多的，如图 9-5 所示。



图 9-5

我们通过解决具体问题，学习一些重要的方法。

（1）创建单位矩阵，比如创建一个 3 行 3 列的单位矩阵，如图 9-6 所示。

（2）创建矩阵。在创建矩阵的专用字符串中，矩阵的行与列之间用分号隔开，行内的元素之间用空格隔开。举例：使用如下的字符换调用 mat()函数创建矩阵，如图 9-7 所示。

同样的，矩阵的方法也是很多的，希望读者对矩阵能特别熟练掌握，毕竟很多数据都可以看作矩阵进行分析，如图 9-8 所示。

```
In  [2]: ret=np.eye(3)
         ret

Out[2]: array([[ 1.,  0.,  0.],
               [ 0.,  1.,  0.],
               [ 0.,  0.,  1.]])
```

图 9-6

```
In  [3]: ret=np.mat("1 2 3;4 5 6;7 8 9")
         ret

Out[3]: matrix([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

图 9-7

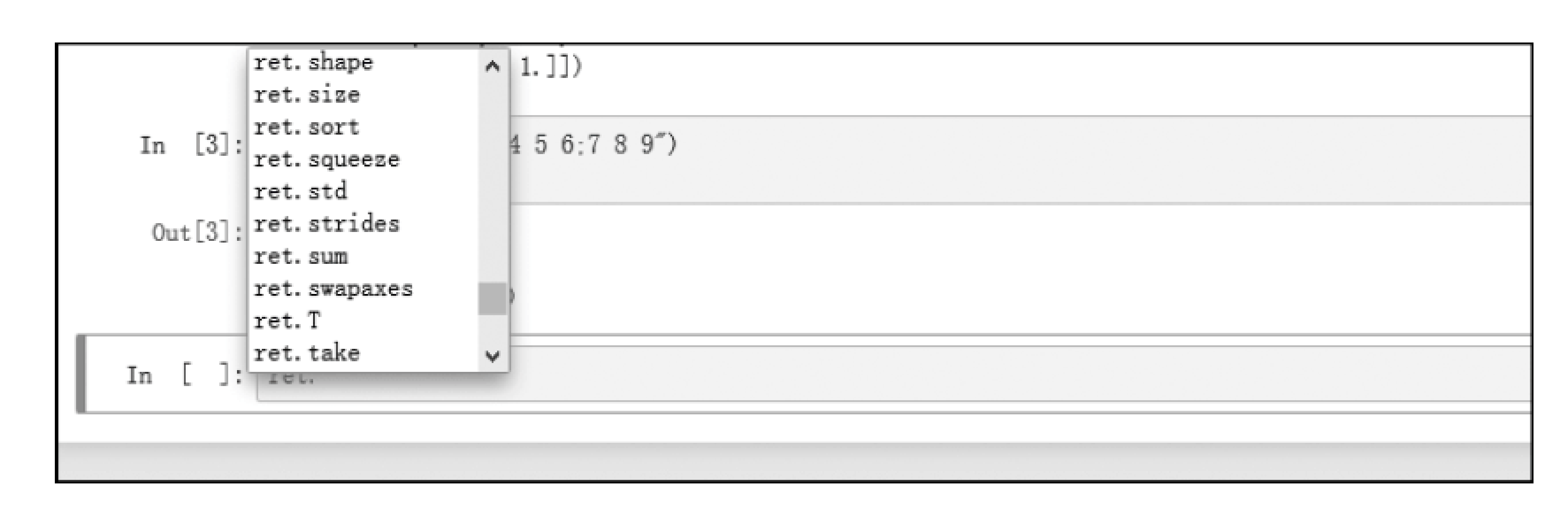


图 9-8

比如,用 T 属性获取转置矩阵,如图 9-9 所示。

```
In  [4]: ret.T

Out[4]: matrix([[1, 4, 7],
                [2, 5, 8],
                [3, 6, 9]])
```

图 9-9

(3) 通过 NumPy 数组创建矩阵,如图 9-10 所示。

```
In  [5]: ret=np.mat(np.arange(16).reshape(4,4))
         ret

Out[5]: matrix([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11],
                [12, 13, 14, 15]])
```

图 9-10

9.1.3 随机数生成

关于随机数的生成方法是读者必须掌握的。NumPy.random 模块对 Python 的内置函数 random 进行了补充,增加了高效生成多种概率分布样本值的函数。举例:我们可以用 normal 得到一个标准正态分布,如图 9-11 所示。

对于随机数的生成,建议读者就用 NumPy 的 random 模块。这里,我们列出常用的

```
In [8]: sam=np.random.normal(size=(4,1))
        sam

Out[8]: array([[-0.86985173],
               [-2.77199762],
               [ 0.27702183],
               [-0.15608159]])

In [9]: sam=np.random.normal(size=(4,4))
        sam

Out[9]: array([[-0.36986124, -1.86823455,  2.42736784, -1.03384532],
               [-0.81237638,  0.80633452, -0.84744915, -0.09427308],
               [ 1.013367  ,  0.58824639,  0.81128371, -0.96527215],
               [ 1.16045793,  0.16483525, -0.33607596,  1.00431376]])
```

图 9-11

统计函数，如表 9-1 所示。

表 9-1

seed	确定随机生成器的种子
permutation	返回一个序列的随机排列
shuffle	对一个序列就地随机排列
rand	产生均匀分布的样本值
randint	从给定的上下限范围内随机选取整数
randn	产生标准的正态分布样本值
binominal	产生二项分布的样本值
normal	产生正态分布的样本值
Beta	产生 Beta 分布的样本值
chisquare	产生卡方分布的样本值
Gamma	产生 Gamma 分布的样本值
uniform	产生在[0,1)中的均匀分布的样本值

下面介绍常见的需要解释的几点：

(1) 伪随机数并不是假随机数，这里的“伪”是有规律的意思，就是计算机产生的伪随机数既是随机的又是有规律的。

(2) 随机种子来自系统时钟，确切地说，是来自计算机主板上的定时/计数器在内存中的记数值。

(3) 随机数是由随机种子根据一定的计算方法计算出来的数值。所以，只要计算方法一定，随机种子一定，那么产生的随机数就不会变。也就是说，伪随机数也是某种对应映射的产物，只不过这个自变量是系统的时间而已。

(4) 如果每次调用 srand()[①]时都提供相同的种子值，那么，你将会得到相同的随机数序列。

9.1.4 Shuffle 和 permutation 的区别

如果传给 permutation 一个矩阵，它将会返回一个打乱后的矩阵副本；而 shuffle 只

① srand()是 C++ 中一个随机数生成种子的函数。

是对这个矩阵进行打乱，无返回值。举例如下：

如图 9-12 所示，就是说对于 shuffle 是没有返回值的，这一点读者需要注意。

```
In  [10]: x=np.arange(10)
          y=np.random.shuffle(x)
          type(y)

Out[10]: NoneType

In  [11]: x=np.arange(10)
          y2=np.random.permutation(x)
          y2

Out[11]: array([0, 5, 1, 9, 7, 8, 4, 3, 2, 6])
```

图 9-12

9.2 矩阵计算

矩阵计算（比如矩阵的加减法、数乘、行列式的计算以及矩阵分解）是任何数组库的重要组成部分，也是实际工作都会用到的。

9.2.1 矩阵的点积运算

如图 9-13 所示。

```
Out[14]: array([[1, 2, 3],
                [4, 5, 6]])

In  [15]: y=np.array([[1,2],[3,4],[5,6]])
          y

Out[15]: array([[1, 2],
                [3, 4],
                [5, 6]])

In  [16]: z=np.dot(x,y)
          z

Out[16]: array([[22, 28],
                [49, 64]])
```

图 9-13

9.2.2 计算特征值

Numpy.linalg 模块中有一组标准的矩阵分解运算以及计算行列式之类的东西。这个跟 MATLAB、R 语言差不多。

下面列举常用的 numpy.linalg 函数，如表 9-2 所示。

表 9-2

diag	按照一维数组的形式返回对角线的元素
dot	矩阵的乘法
trace	计算对角线的和

续表

det	计算矩阵的行列式
eig	计算方阵的特征值和特征向量
inv	计算方阵的逆
solve	解线性方程组 $Ax=b$,其中 A 为一个方阵
lstsq	计算 $Ax=b$ 的最小二乘解
qr	计算 QR 分解
svd	计算奇异值分解(SVD)

9.3 蒙特卡洛模拟

蒙特卡洛模拟是金融行业和数据分析科学中都会用到的算法之一,也是非常重要的算法。之所以重要,是因为比如在期权定价或风险管理问题上都有很强的能力。和其他的数值方法相比,蒙特卡洛方法很容易处理高维问题,而且这种问题的复杂度和计算需求通常是按照线性方式增长的,如图 9-14 所示。

```
In [ ]: import numpy as np
        def Monte_Pi(num):
            count=0
            for i in np.arange(1,num+1):
                X=np.random.uniform(0,1)
                Y=np.random.uniform(0,1)
                if(X**2+Y**2<1):
                    count=count+1
            return 4*count/num

In [3]: Monte_Pi(100000)
Out[3]: 3.13688

In [4]: Monte_Pi(1000000)
Out[4]: 3.13994

In [5]: Monte_Pi(10000000)
Out[5]: 3.1419964
```

图 9-14

当然,蒙特卡洛的缺点也是很明显的,本身是高计算需求的,即使是一个简单的问题也需要海量的计算。因此,尽可能设计高效的蒙特卡洛算法。

第 10 章

Pandas 介绍

10.1 Pandas 的特点

Pandas 是金融行业从业人员做数据分析的首选库，如无必要，建议尽可能使用 Pandas，这也是本书前面几个部分对 Python 介绍得并不多的原因。

Pandas 使得数据分析工作变得更加简单。

Pandas 是基于 NumPy 构建的，这让以 NumPy 为中心的应用变得更加简单好用。

Pandas 是 Wes McKinney 在一家对冲基金公司做量化投资工作时创建的，这让我们更有理由用好这个库。

10.2 Pandas 的数据结构

要使用 Pandas，其实最需要熟悉的就是两个数据结构：Series 和 DataFrame。这两个数据结构能够让我们解决金融行业的绝大部分的金融数据。

这里有必要再讲讲为什么可以这么理解。其实我们看到的结构化数据基本上就可以理解为一个矩阵。Series 就是一个向量，DataFrame 就是一个矩阵，向量也是特殊的矩阵，N 行 1 列的一种矩阵。

大家可能也会发现这和 R 语言是很类似的，同时我们还会进一步讲解为什么有了 Python 的 list，进一步有了 NumPy 的 array，为什么现在还需要有个 DataFrame。

10.2.1 Series

Series 是类似于一维数组的对象。它由一组数据以及一组相关的数据标签，其实就是索引组成的。举例如下：

Series 的形式如图 10-1 所示，左边是索引，右边是值。由于我们没有为数据制定索引，于是系统会自动创建一个 0～(N－1)(N 为数据的个数)的整数型索引。我们可以通过 Series 的 value 和 index 属性获取，如图 10-2 所示。

```
In  [22]: import pandas as pd
          ret=pd.Series([11.45,2.67,36.45,0.4])
          ret

Out[22]: 0    11.45
         1     2.67
         2    36.45
         3     0.40
         dtype: float64
```

图 10-1

```
In  [23]: ret.values
Out[23]: array([ 11.45,    2.67,    36.45,    0.4 ])

In  [24]: ret.index
Out[24]: RangeIndex(start=0, stop=4, step=1)
```

图 10-2

我们也可以自己进行设置，创建一个对数据进行标记的索引。

举例如图 10-3 所示：

```
In  [25]: stockdata=pd.Series([3246.22,3255.78,3229.13,3245.22],index=["open","high","low","close"])
          stockdata
Out[25]: open     3246.22
         high     3255.78
         low      3229.13
         close    3245.22
         dtype: float64
```

图 10-3

可以看出，这和 NumPy 相比，可以通过索引的方式选取 Series 的单个或一组值。如图 10-4 所示。

```
In  [26]: stockdata["open"]
Out[26]: 3246.2199999999998

In  [27]: stockdata[["high","close"]]
Out[27]: high     3255.78
         close    3245.22
         dtype: float64
```

图 10-4

我们还可以把 Series 看作是一个有定长的有序字典，因为它本身就是一个索引值到数据值的映射。

从图 10-5 可以看出，Python 语言的互动性很强，而且只要读者对这门语言感兴趣，就可以很自然地猜到怎么去写程序，不断试错，并且还可以百度。

```
In  [28]: "volume" in stockdata
Out[28]: False
```

图 10-5

最后，如果数据被存放在一个 Python 字典中，也是可以直接通过这个字典创建 Series，如图 10-6 所示。

```
In  [29]: sdata={"open":3246,"high":3255,"low":3229,"close":3245}
          ret=pd.Series(sdata)
          ret
Out[29]: close    3245
         high     3255
         low      3229
         open     3246
         dtype: int64
```

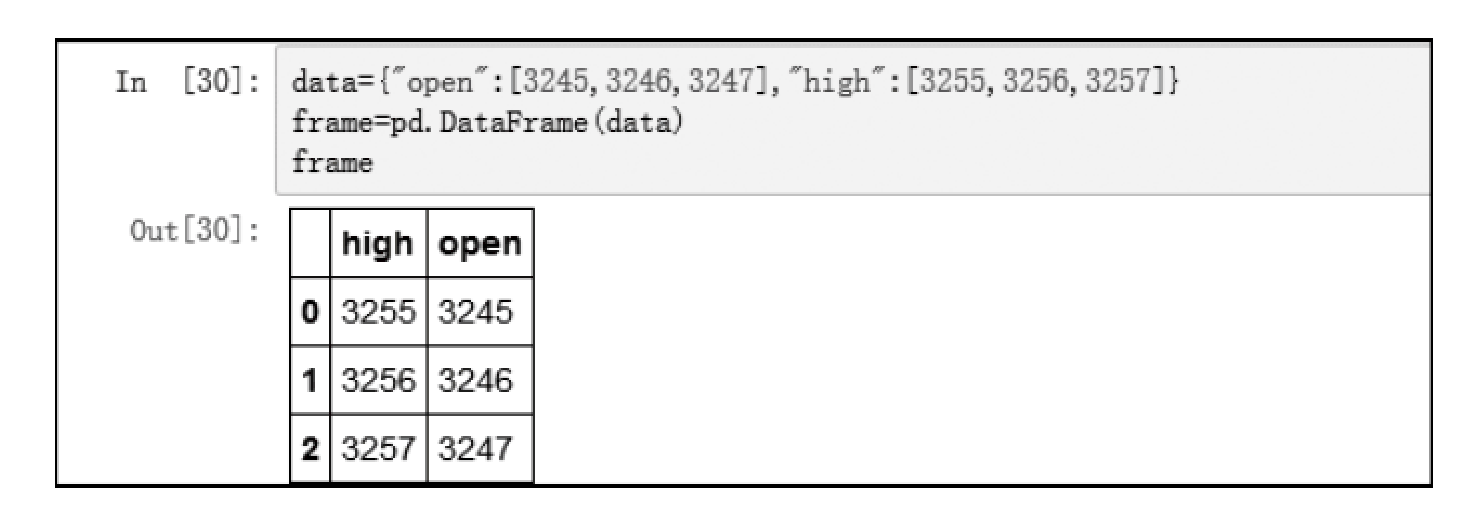
图 10-6

10.2.2 DataFrame

DataFrame 是一个表格型的数据结构。它含有一组有序的列，每列可以是不同的值类型(数值、字符串、布尔值等)。DataFrame 既包含行索引，也包含列索引，也可以看作是由 Series 组成的字典。

当然，在这里读者可能会想到 Numpy 的多维数组，其实虽然 DataFrame 是按照二维数据结构保存的，但还是一样可以轻松地表示为更高维度的数据。这里，我们主要是面向金融行业的金融数据进行分析。

创建 DataFrame 的办法很多，最常用的是直接传入一个等长的列表或 NumPy 数组组成的字典，如图 10-7 所示。



```
In  [30]: data={"open":[3245,3246,3247],"high":[3255,3256,3257]}
          frame=pd.DataFrame(data)
          frame
Out[30]:
```

	high	open
0	3255	3245
1	3256	3246
2	3257	3247

图 10-7

跟 Series 一样，DataFrame 会自动加上索引，且全部列会有序排列。

10.2.3 重要的方法介绍

1. 重新索引

Pandas 对象的一个重要方法是 reindex，作用就是创建一个适应新索引的对象。比如图 10-8 所示。

```
In  [32]: sdata={"open":3246,"high":3255,"low":3229,"close":3245}
          ret=pd.Series(sdata)
          ret.reindex(["close","open","low","high"])

Out[32]: close    3245
         open     3246
         low      3229
         high     3255
         dtype: int64
```

图 10-8

2. 数据填充方法

对于金融时间序列，重新索引可能需要对一些数据进行差值处理，这个对很多做高频交易的可能很有必要，这涉及怎么处理分笔数据，而分笔数据可能会有很多是缺失数据，这就需要按照某种规则或者算法进行填充补全。

上面的那个例子，如我们输入的某一个索引值不存在就引入缺失值，如图 10-9 所示。

第一种方法是因为索引值不存在，设置为 NaN；第二种方法指定为填充值为 0，如图 10-10 所示。

```
In  [33]: ret.reindex(["close","open","low","high","ma5"])

Out[33]: close    3245.0
         open     3246.0
         low      3229.0
         high     3255.0
         ma5         NaN
         dtype: float64

In  [34]: ret.reindex(["close","open","low","high","ma5"],fill_value=0)

Out[34]: close    3245
         open     3246
         low      3229
         high     3255
         ma5         0
         dtype: int64
```

图 10-9

```
In  [36]: ret.reindex(["close","open","ma5","low","high"],method="ffill")

Out[36]: close    3245
         open     3246
         ma5      3229
         low      3229
         high     3255
         dtype: int64

In  [37]: ret.reindex(["close","open","ma5","low","high"],method="bfill")

Out[37]: close    3245
         open     3246
         ma5      3246
         low      3229
         high     3255
         dtype: int64
```

图 10-10

这里我们列出 reindex 差值方法的各个选项，如表 10-1 所示。

表 10-1

参数	说　明
ffill	前向填充
bfill	后向填充

3. 删除指定项

删除 Series 中的一个数或者多个项，只要有一个索引或者列表即可，这里需要注意的是返回的是删除指定项后的新的对象。我们可以理解为删除矩阵的某一个元素或者某几行某几列，如图 10-11 所示。

这里读者需要注意的是：我们使用的是 drop 方法，而不是 delete，也不是 remove。

对于 DataFrame，可以删除任意一个元素或者任意行或任意列。

例如，删除某一行，如图 10-12 所示。

例如，删除某一列，如图 10-13 所示。

这里需要注意的是，删除某一列的时候，要指定 axis=1，就是说默认的是针对的行索引进行操作的，如果改成针对列索引，设置 axis=1 即可。

```
In  [1]: import numpy as np
         import pandas as pd
         data=pd.Series(np.arange(5),index=["open","high","low","close","ma5"])
         ndata=data.drop("high")
         ndata

Out[1]: open     0
        low      2
        close    3
        ma5      4
        dtype: int32
```

图 10-11

```
In  [2]: data=pd.DataFrame(np.arange(16).reshape(4,4),index=["one","two","three","four"]
                          ,columns=["open","high","low","close"])
         ndata1=data.drop(["one"])
         ndata1

Out[2]:
```

	open	high	low	close
two	4	5	6	7
three	8	9	10	11
four	12	13	14	15

图 10-12

```
In  [3]: data=pd.DataFrame(np.arange(16).reshape(4,4),index=["one","two","three","four"]
                          ,columns=["open","high","low","close"])
         ndata2=data.drop(["open"],axis=1)
         ndata2

Out[3]:
```

	high	low	close
one	1	2	3
two	5	6	7
three	9	10	11
four	13	14	15

图 10-13

4. 索引、选取和过滤

这里我们进一步系统地介绍一下查看数据的方法，主要包括索引、选取和过滤。

Series 的索引既可以按照 NumPy 数组的方式，也可以根据 index 的方式，如图 10-14 所示。

```
In  [4]: data=pd.Series(np.arange(5),index=["open","high","low","close","ma5"])
         data[1]

Out[4]: 1

In  [5]: data["high"]

Out[5]: 1
```

图 10-14

不过，这里最需要注意的是标签的切片与普通的 Python 切片运算不同，右边是包含的，也就是说是一个闭区间的，如图 10-15 所示。

```
In  [6]: data=pd.Series(np.arange(5),index=["open","high","low","close","ma5"])
         data["high":"close"]

Out[6]: high     1
        low      2
        close    3
        dtype: int32

In  [7]: data[1:3]

Out[7]: high     1
        low      2
        dtype: int32
```

图 10-15

当然，进行更改也是简单的，如图 10-16 所示。

```
In  [8]: data["high":"close"]=[3255,3245,3240]

In  [9]: data

Out[9]: open        0
        high     3255
        low      3245
        close    3240
        ma5         4
        dtype: int32
```

图 10-16

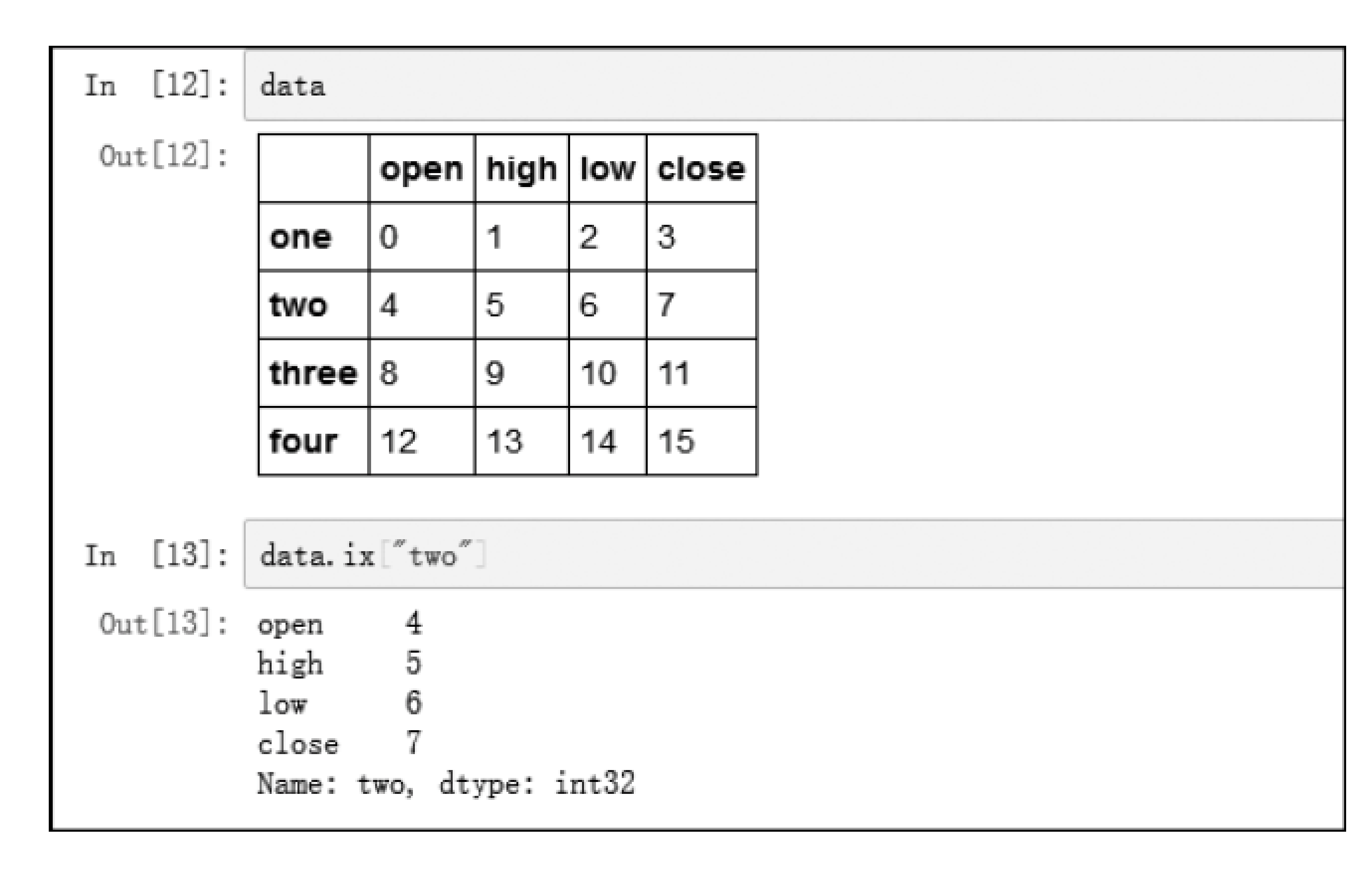
DataFrame 进行索引也就是获取一个或者多个列，如图 10-17 所示。

```
In  [11]: data=pd.DataFrame(np.arange(16).reshape(4,4),index=["one","two","three","four"]
                            ,columns=["open","high","low","close"])
          data["high"]

Out[11]: one       1
         two       5
         three     9
         four     13
         Name: high, dtype: int32
```

图 10-17

但是，这里需要注意的是对行的索引我们需要用 ix 方法，如图 10-18 所示。



```
In  [12]: data

Out[12]:
```

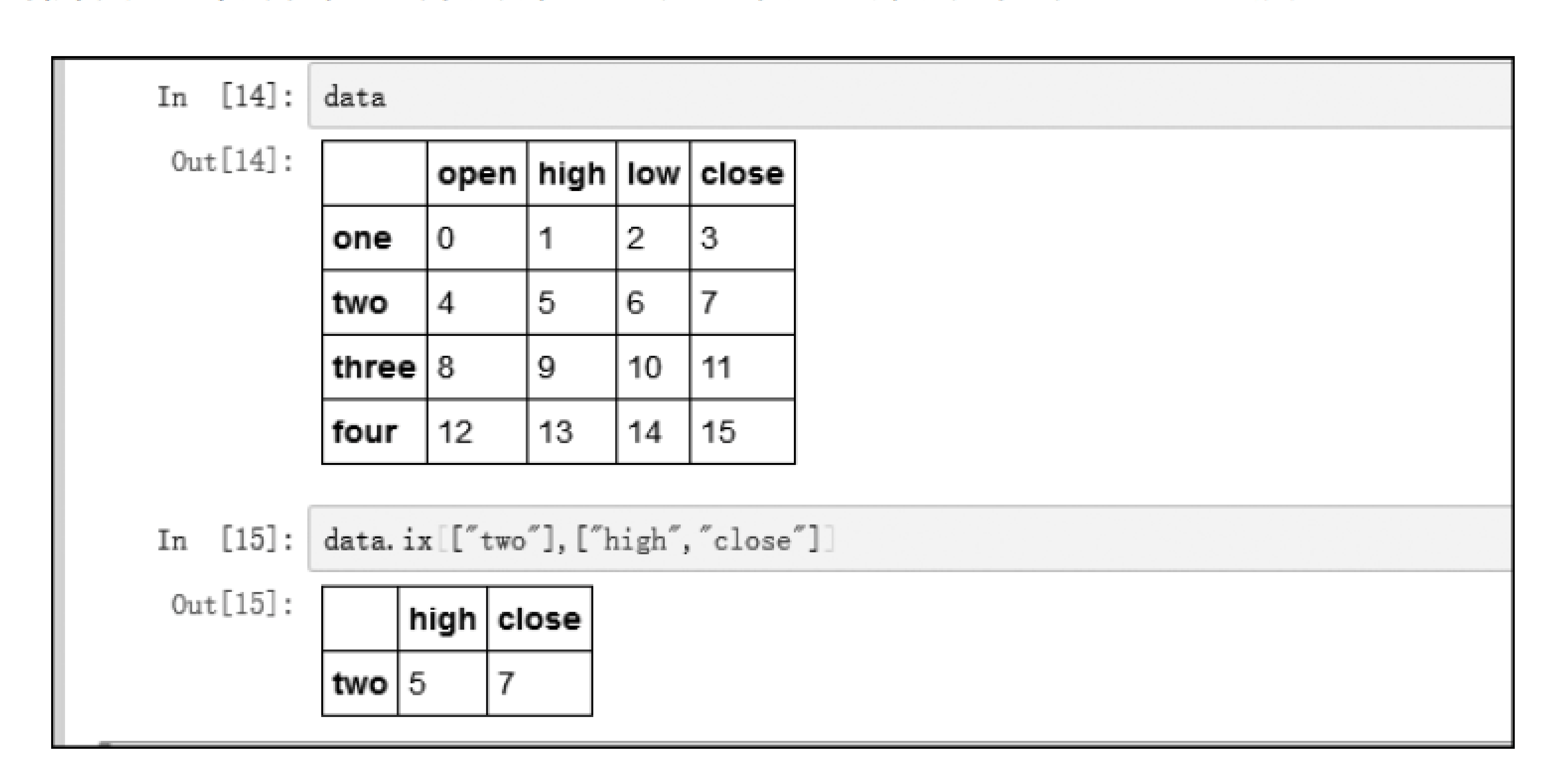
	open	high	low	close
one	0	1	2	3
two	4	5	6	7
three	8	9	10	11
four	12	13	14	15

```
In  [13]: data.ix["two"]

Out[13]: open     4
         high     5
         low      6
         close    7
         Name: two, dtype: int32
```

图 10-18

如果我们需要索引某一行的某几列,也是一样的,如图 10-19 所示。



```
In [14]: data
```

Out[14]:

	open	high	low	close
one	0	1	2	3
two	4	5	6	7
three	8	9	10	11
four	12	13	14	15

```
In [15]: data.ix[["two"],["high","close"]]
```

Out[15]:

	high	close
two	5	7

图　10-19

5. 一些特殊的方法。

(1) 通过切片方法选取行,如图 10-20 所示。

```
In [17]: data
```

Out[17]:

	open	high	low	close
one	0	1	2	3
two	4	5	6	7
three	8	9	10	11
four	12	13	14	15

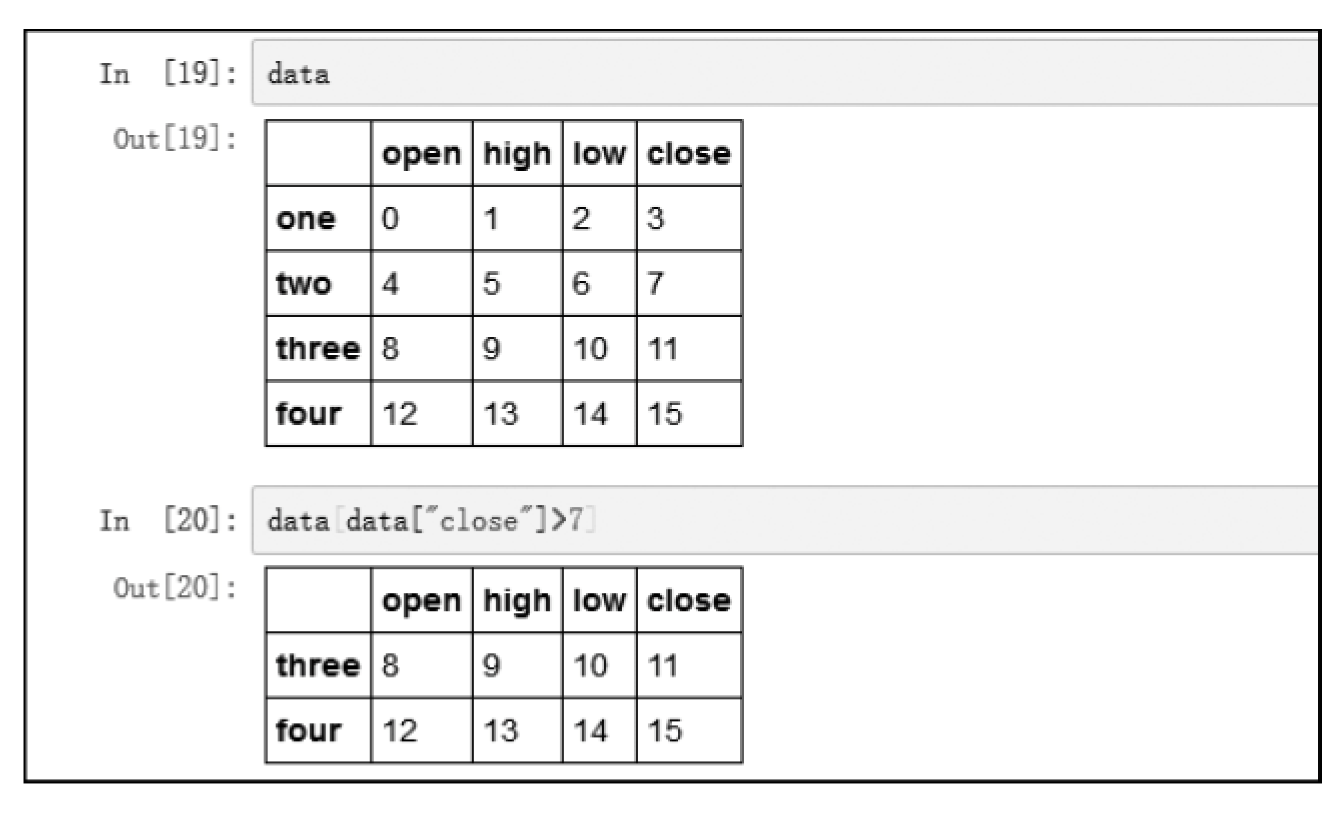
```
In [18]: data[:3]
```

Out[18]:

	open	high	low	close
one	0	1	2	3
two	4	5	6	7
three	8	9	10	11

图　10-20

(2) 布尔型选取行,如图 10-21 所示。



```
In [19]: data
```

Out[19]:

	open	high	low	close
one	0	1	2	3
two	4	5	6	7
three	8	9	10	11
four	12	13	14	15

```
In [20]: data[data["close"]>7]
```

Out[20]:

	open	high	low	close
three	8	9	10	11
four	12	13	14	15

图　10-21

因为对 Python 来说，一切都是对象。所以同类的就是可以比较的，比如，我们可以按图 10-22 操作：

In [21]:
```
data
```

Out[21]:

	open	high	low	close
one	0	1	2	3
two	4	5	6	7
three	8	9	10	11
four	12	13	14	15

In [22]:
```
data<5
```

Out[22]:

	open	high	low	close
one	True	True	True	True
two	True	False	False	False
three	False	False	False	False
four	False	False	False	False

图 10-22

对筛选的数据进行赋值也是简单的，如图 10-23 所示。

In [23]:
```
data
```

Out[23]:

	open	high	low	close
one	0	1	2	3
two	4	5	6	7
three	8	9	10	11
four	12	13	14	15

In [25]:
```
data[data<5]=100
data
```

Out[25]:

	open	high	low	close
one	100	100	100	100
two	100	5	6	7
three	8	9	10	11
four	12	13	14	15

图 10-23

索引、选取和过滤的方法总结如表 10-2 所示。

表 10-2

类　型	说　明
obj[val]	选取 DataFrame 的单个列或一组列，以及针对一些方法：布尔型(过滤行)、切片(过滤行)
obj. ix[val]	选取行或者列。obj[:,val]选取单个列或多个列；obj[val1,val2]同时选取行和列
reindex	将一个或多个列标签匹配到新索引

6. 算数运算和数据对齐

算数运算，举例如图 10-24 所示。

```
In  [26]: data1=pd.Series([3245,3255,3229,3245],index=["open","high","low","close"])
          data1

Out[26]: open     3245
         high     3255
         low      3229
         close    3245
         dtype: int64

In  [27]: data2=pd.Series(np.arange(5),index=["open","high","low","close","ma5"])
          data2

Out[27]: open     0
         high     1
         low      2
         close    3
         ma5      4
         dtype: int32

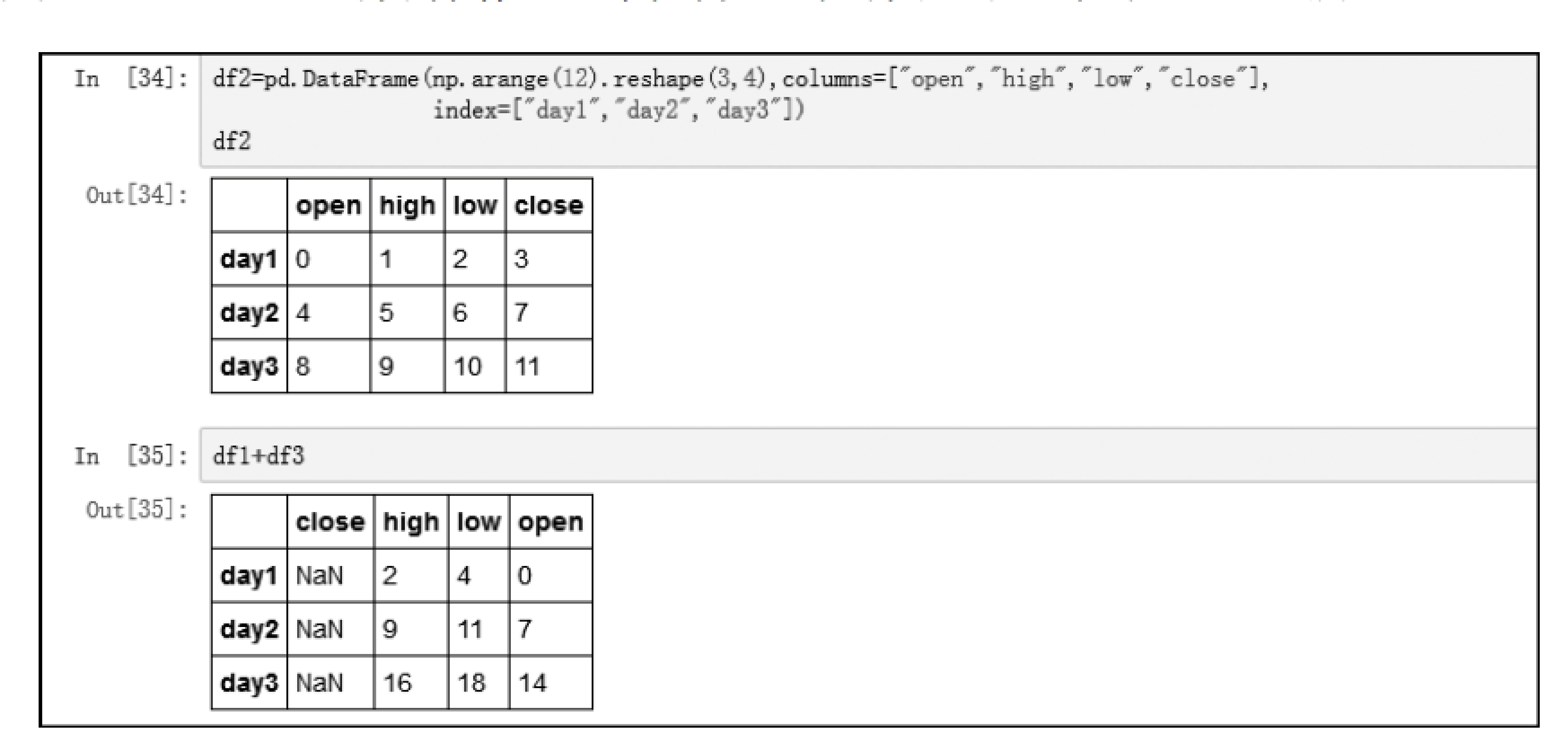
In  [28]: data1+data2

Out[28]: close    3248.0
         high     3256.0
         low      3231.0
         ma5         NaN
         open     3245.0
         dtype: float64
```

图 10-24

在这里，读者可以看到自动的数据对齐操作在不重叠的索引处引入了 NaN 值。缺失值会在算术运算中保持和传递。

对于 DataFrame，对齐操作可以同时发生在行和列上，如图 10-25 所示。



```
In  [34]: df2=pd.DataFrame(np.arange(12).reshape(3,4),columns=["open","high","low","close"],
                           index=["day1","day2","day3"])
          df2
```

Out[34]:

	open	high	low	close
day1	0	1	2	3
day2	4	5	6	7
day3	8	9	10	11

```
In  [35]: df1+df3
```

Out[35]:

	close	high	low	open
day1	NaN	2	4	0
day2	NaN	9	11	7
day3	NaN	16	18	14

图 10-25

大家看到的，没有重叠的相加的就是 NaN。

我们可以在 Series 或者 DataFrame 重新索引时，重新指定一个填充值，如图 10-26 所示。

```
In  [36]: df1.add(df3,fill_value=10000)
```

Out[36]:

	close	high	low	open
day1	10003.0	2	4	0
day2	10007.0	9	11	7
day3	10011.0	16	18	14

图 10-26

Pandas 中的算术方法总结如表 10-3 所示。

表 10-3

方法	说　明
add	用于加法(+)
sub	用于减法(-)
div	用于除法(/)
mul	用于乘法(*)

10.2.4 DataFrame 和 Series 之间的运算

1. 加减法

先看例子,如图 10-27 所示。

```
In [37]: df1=pd.DataFrame(np.arange(9).reshape(3,3),columns=["open","high","low"],
                          index=["day1","day2","day3"])
         df1
```

Out[37]:

	open	high	low
day1	0	1	2
day2	3	4	5
day3	6	7	8

```
In [38]: df1+8888
```

Out[38]:

	open	high	low
day1	8888	8889	8890
day2	8891	8892	8893
day3	8894	8895	8896

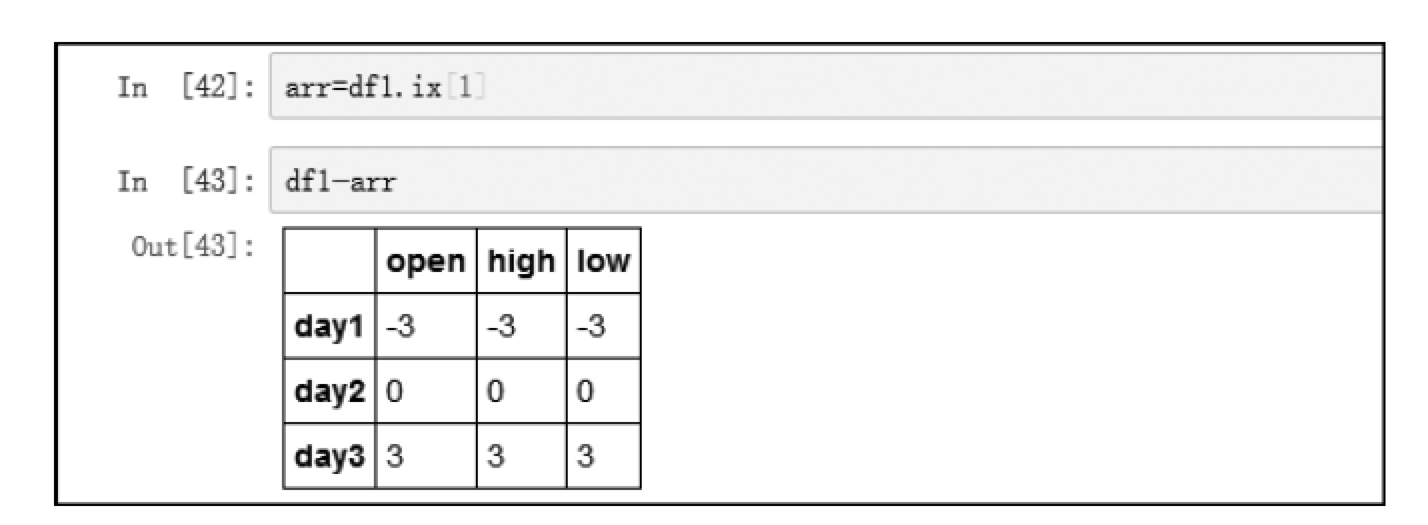
```
In [39]: df1*10000
```

Out[39]:

	open	high	low
day1	0	10000	20000
day2	30000	40000	50000
day3	60000	70000	80000

图 10-27

还是一句话,在 Python 中,一切都是对象,那么 DataFrame 和 Series 的加减法也就很容易理解了,如图 10-28 所示。



```
In [42]: arr=df1.ix[1]

In [43]: df1-arr
```

Out[43]:

	open	high	low
day1	-3	-3	-3
day2	0	0	0
day3	3	3	3

图 10-28

2. 顺序和排名

要对行或者列进行排序，可以使用 sort_index 方法。返回的是一个重新排序的新对象，如图 10-29 所示。

```
In  [44]: data=pd.Series(np.arange(5),index=["open","high","low","close","ma5"])
          data.sort_index()

Out[44]: close    3
         high     1
         low      2
         ma5      4
         open     0
         dtype: int32
```

图　10-29

从图 10-29 可以看出，默认的排序规则是升序排列的，这里根据的是第一个字符，是升序排列的。当然，我们也可以设置为降序排列，如图 10-30 所示。

```
In  [46]: data.sort_index(ascending=False)

Out[46]: open     0
         ma5      4
         low      2
         high     1
         close    3
         dtype: int32
```

图　10-30

同样的，我们可以针对值进行排序，如图 10-31 所示。

```
In  [53]: data=pd.Series([1000,7,-56,22,6666],index=["open","high","low","close","ma5"])
          data.sort_values()

Out[53]: low       -56
         high        7
         close      22
         open     1000
         ma5      6666
         dtype: int64

In  [54]: data.sort_values(ascending=False)

Out[54]: ma5      6666
         open     1000
         close      22
         high        7
         low       -56
         dtype: int64
```

图　10-31

对于 DataFrame，可以根据任意的列上的索引进行排序，如图 10-32 所示。

```
In  [56]: df1=pd.DataFrame(np.arange(8).reshape(2,4),index=["Thu","Fri"],columns=["open","high","low","close"])
          df1.sort_index()

Out[56]:
```

	open	high	low	close
Fri	4	5	6	7
Thu	0	1	2	3

```
In  [57]: df1.sort_index(axis=1)

Out[57]:
```

	close	high	low	open
Thu	3	1	2	0
Fri	7	5	6	4

图　10-32

3. 常用统计

我们可以用 describe 方法查询对象的统计特征，如图 10-33 所示。

```
In  [61]: df1=pd.DataFrame(np.arange(8).reshape(2,4),index=["Thu","Fri"],columns=["open","high","low","close"])
          df1
```

Out[61]:

	open	high	low	close
Thu	0	1	2	3
Fri	4	5	6	7

```
In  [62]: df1.describe()
```

Out[62]:

	open	high	low	close
count	2.000000	2.000000	2.000000	2.000000
mean	2.000000	3.000000	4.000000	5.000000
std	2.828427	2.828427	2.828427	2.828427
min	0.000000	1.000000	2.000000	3.000000
25%	1.000000	2.000000	3.000000	4.000000
50%	2.000000	3.000000	4.000000	5.000000
75%	3.000000	4.000000	5.000000	6.000000
max	4.000000	5.000000	6.000000	7.000000

图 10-33

我们列出所有与描述统计相关的方法，如表 10-4 所示。

表 10-4

count	非 NaN 值的统计
describe	统计汇总
min，max	计算最小值，最大值
argmin，argmax	计算最小值，最大值的索引所在的位置
idxmin，idxmax	计算最小值，最大值的索引
quantile	计算样本的分位数
sum	样本值的总和
mean	样本值的均值
median	样本值的算数中位数
mad	根据平均值计算平均绝对离差
var	样本值的方差
std	样本值的标准差
skew	样本值的偏度（三阶矩）
kurt	样本值的峰度（四阶矩）
cumsum	样本值的累计和
cummin，cummax	样本值的累计最小值，最大值
cumprod	样本值的累计积
diff	计算一阶差分
pct_change	计算百分数变化

4. 处理缺失数据

缺失数据在大部分数据分析中都会用到，Pandas 在这方面的处理是最方便的。Pandas 使用浮点值 NaN(Not a Number)表示浮点和非浮点数组中的缺失数据。它

只是一个便于被检测出来的标记而已，如图 10-34 所示。

```
In  [2]: import numpy as np
         import pandas as pd
         data=pd.Series([1000,"open","hello",np.nan,56])
         data

Out[2]: 0     1000
        1     open
        2    hello
        3      NaN
        4       56
        dtype: object
```

图 10-34

Python 本身的 None 值也会被当作 NaN 处理的。

这就涉及怎么判断是否是一个缺失值，我们使用函数 isnull，如图 10-35 所示。

```
In  [3]: data[2]=None

In  [4]: data

Out[4]: 0    1000
        1    open
        2    None
        3     NaN
        4      56
        dtype: object

In  [5]: data.isnull()

Out[5]: 0    False
        1    False
        2     True
        3     True
        4    False
        dtype: bool
```

图 10-35

5. 过滤缺失数据

过滤缺失值数据，我们使用 dropna。对于 Series，dropna 返回一个仅含非空数据和索引值的 Series，如图 10-36 所示。

```
In  [5]: data.isnull()

Out[5]: 0    False
        1    False
        2     True
        3     True
        4    False
        dtype: bool

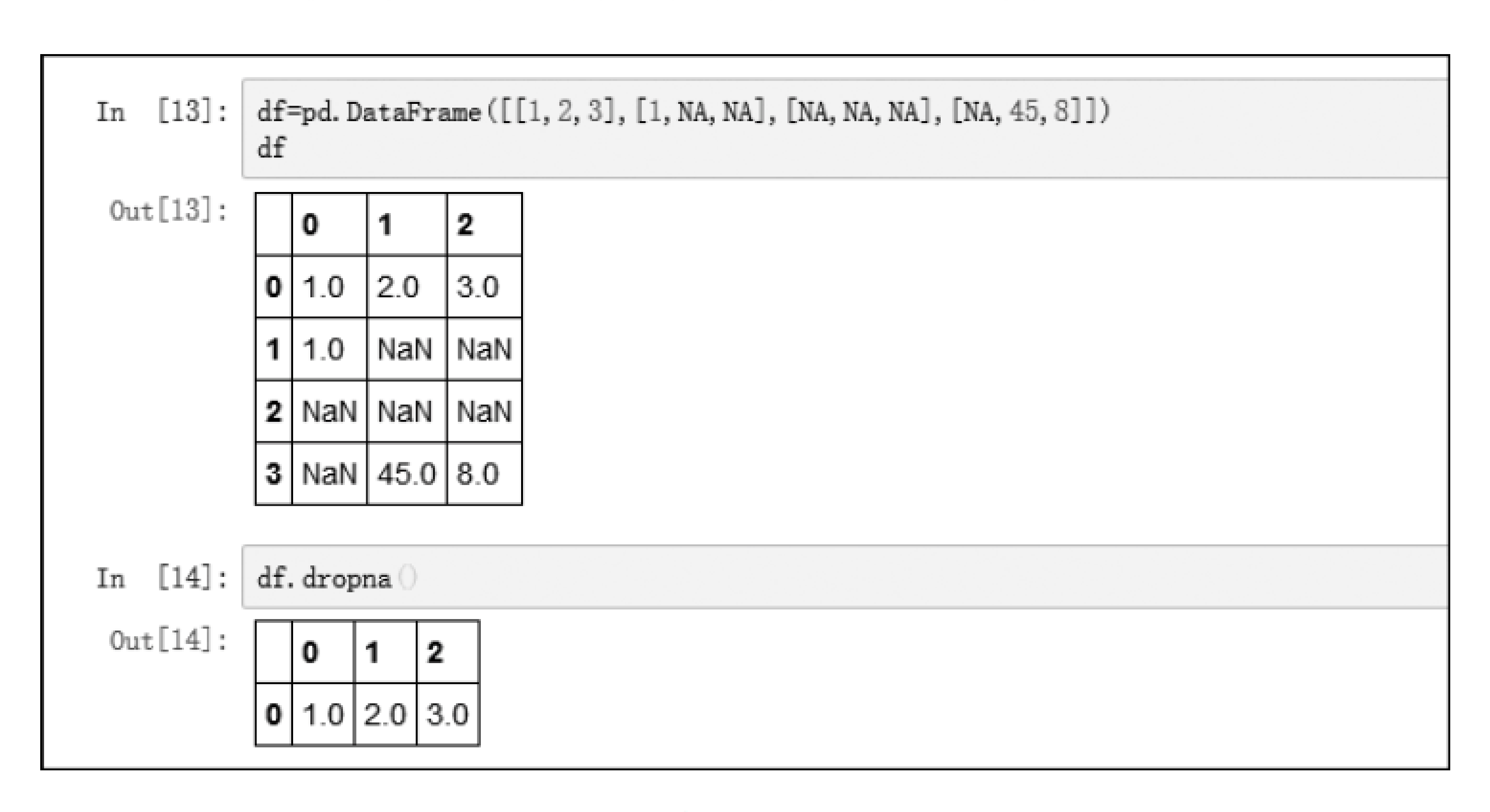
In  [6]: data.dropna()

Out[6]: 0    1000
        1    open
        4      56
        dtype: object
```

图 10-36

对于 DataFrame 可能会比较复杂一些，要是我们想得到，可以删除 NaN 但要考虑删除 NaN 所占的行还是列。

如图 10-37 所示，默认的是删除包含缺失数据所在的行。



In [13]:
```
df=pd.DataFrame([[1,2,3],[1,NA,NA],[NA,NA,NA],[NA,45,8]])
df
```

Out[13]:

	0	1	2
0	1.0	2.0	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	45.0	8.0

In [14]:
```
df.dropna()
```

Out[14]:

	0	1	2
0	1.0	2.0	3.0

图 10-37

如果我们至少需要删除某一行全是 NaN 的，我们需要指定参数为 how＝"all"。如图 10-38 所示。

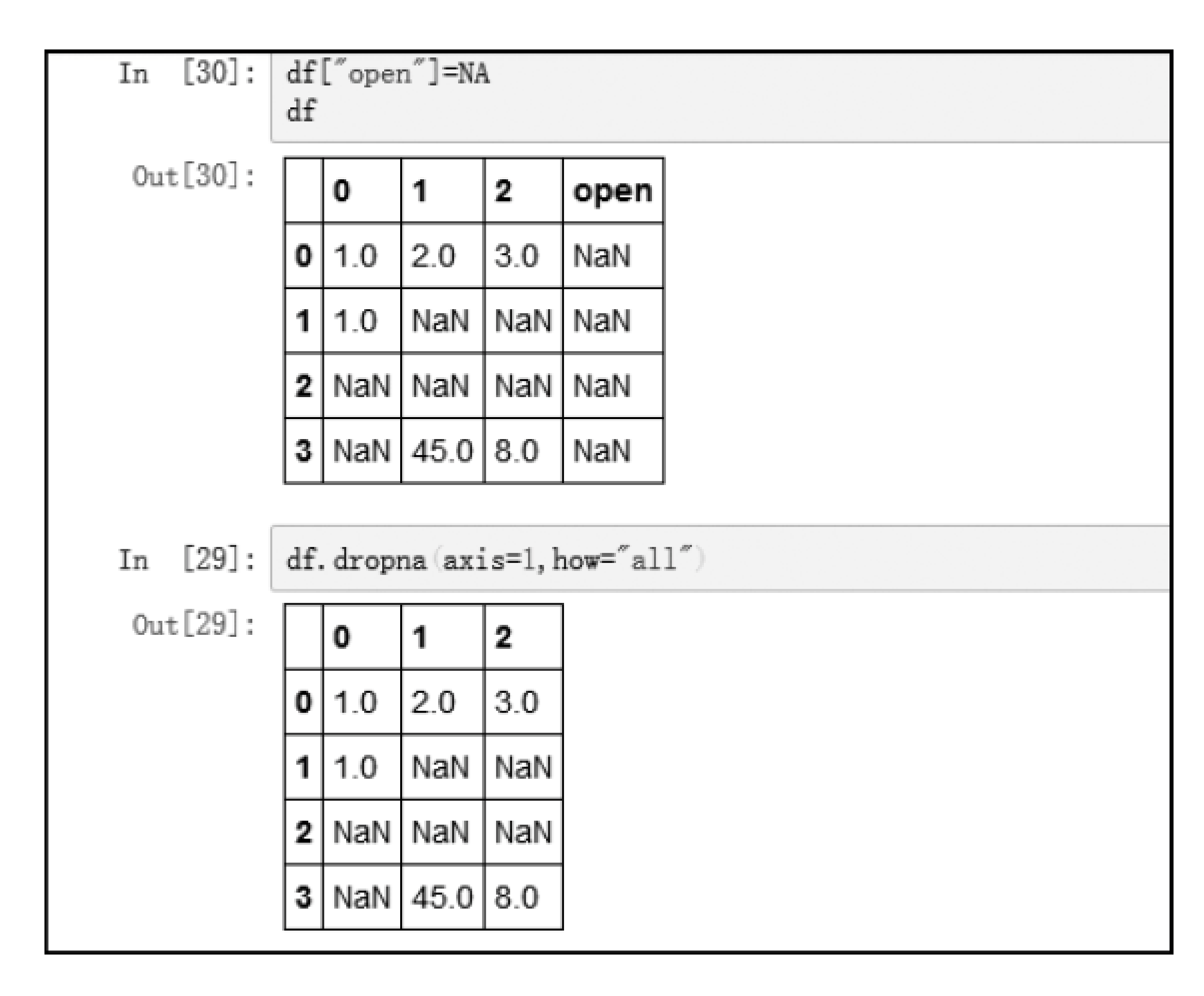
In [17]:
```
df.dropna(how="all")
```

Out[17]:

	0	1	2
0	1.0	2.0	3.0
1	1.0	NaN	NaN
3	NaN	45.0	8.0

图 10-38

如果需要删除 NaN 所在的列，需要在参数中设置 axis＝1，如图 10-39 所示。



In [30]:
```
df["open"]=NA
df
```

Out[30]:

	0	1	2	open
0	1.0	2.0	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	45.0	8.0	NaN

In [29]:
```
df.dropna(axis=1,how="all")
```

Out[29]:

	0	1	2
0	1.0	2.0	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	45.0	8.0

图 10-39

6. 填充缺失数据

对于填充缺失数据也是必须的，我们使用 fillna 函数。通过 fillna 函数，我们把缺失值换成需要替换掉的那个常数值，如图 10-40 所示。

```
In  [47]: df.fillna(100000)
```

Out[47]:

	0	1	2	open
0	1.0	2.0	3.0	100000.0
1	1.0	100000.0	100000.0	100000.0
2	100000.0	100000.0	100000.0	100000.0
3	100000.0	45.0	8.0	100000.0

图　10-40

7. 总结

缺失值的处理方法如表 10-5 所示。

表　10-5

dropna	根据行列标签中存在的缺失数据进行过滤
filla	用指定值（或者差值方法，比如前向 ffill 或者后向 bfill）填充缺失数据
isnull	返回一个含有布尔型的对象，这些布尔值可表示哪些是缺失值，哪些不是



Pandas 读取数据和规整化

11.1 读取数据

我们这章介绍读取本地数据的用法，主要是包括 txt、csv 格式的文本数据。

这里我们主要集中介绍如何使用 read_csv 和 read_table。

1. 使用 read_csv 函数

假设我们在计算机 G 盘 pybook 文件下面有这样一个 txt 文档，如图 11-1 所示。

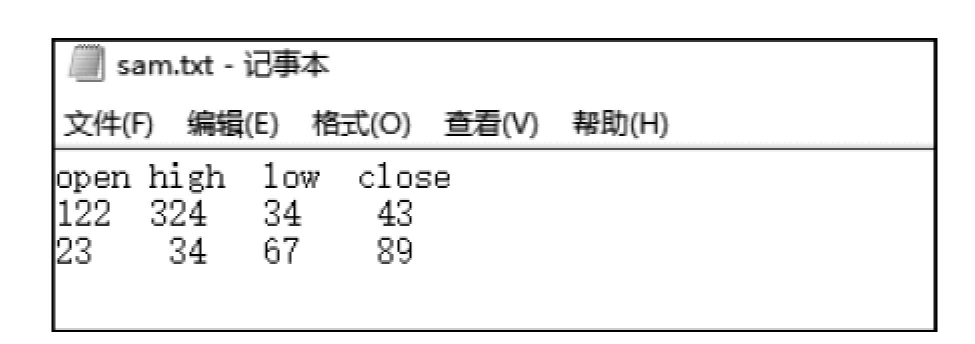


图 11-1

当然我们也可以直接通过 python 语句进行查看，如图 11-2 所示。

```
In [56]: list(open("G:\pybook\sam.txt"))

Out[56]: ['open high  low  close\n', '122  324  34    43\n', '23    34   67    89']
```

图 11-2

那么我们用该函数打开，如下图 11-3 所示。

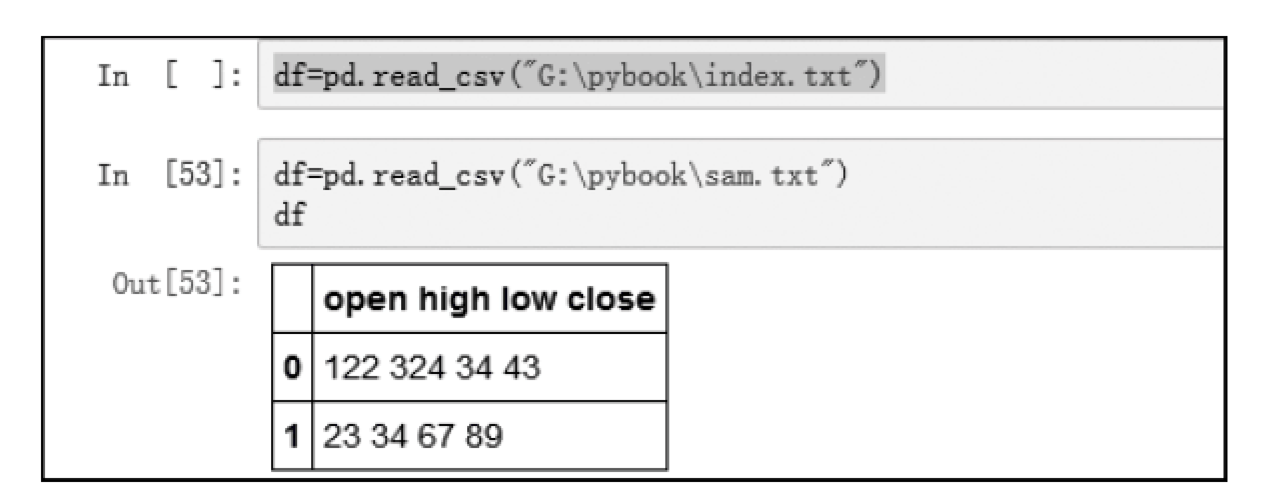


图 11-3

但是，如果我们遇到有中文字符的文本，又会遇到怎样的问题？同样地，我们在计算机 G 盘 pybook 文件下面有这样一个 data2.txt 文档。

那么，我们按照上面的方法打开，就会遇到这样的问题，如图 11-4 所示。

我们不需要去关注以上提示，问题在于如何解决中文字符的问题。

2. 使用 Notepad ++

怎么解决这个问题呢？可能百度也不大容易去解决好这个问题，这里告诉大家一个

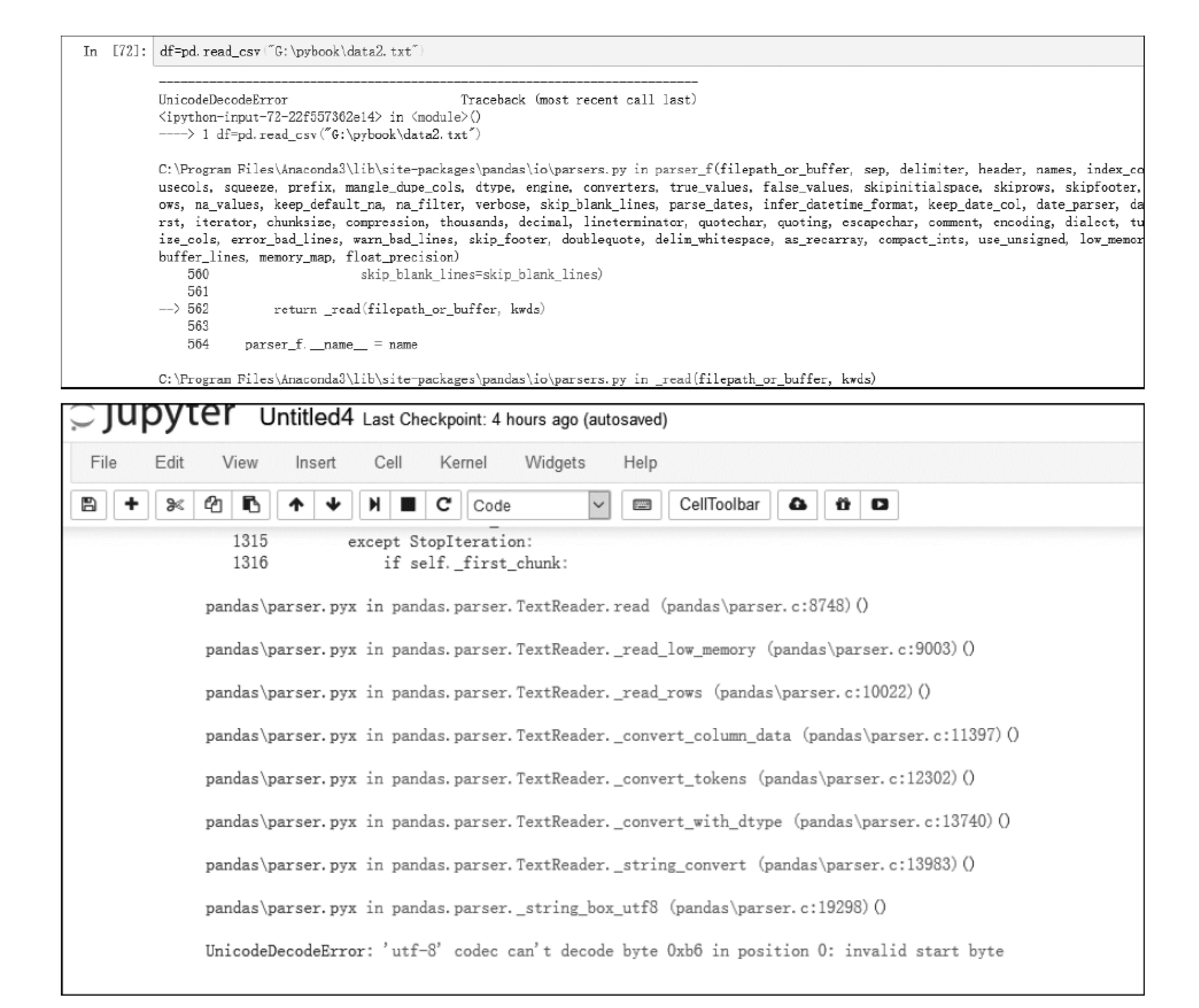


图 11-4

最简单直接的办法。那就是通过 Notepad ++ 来处理这个问题。读者可以自行百度进行下载安装。

Notepad ++ 是 Windows 操作系统下的一套文本编辑器。Notepad ++ 功能比 Windows 中的 Notepad(记事本)强大,除了可以用来制作一般的纯文字说明文件外,还适合编写计算机程序代码。

第一步,我们用 Notepad ++ 打开该文件。

第二步,选择格式——转为 utf-8 编码。

第三步,单击保存。

接下来,我们继续操作。

这里,我们把 txt 文件换成 csv 文件,如果遇到有中文字符,按照上面的方式,用 Notepad ++ 处理一下就可以解决问题了,如图 11-5 所示。

3. 使用 read_table 函数

如图 11-6 所示。

我们可以用 head 方法查看数据的前 5 行,如图 11-7 所示。

细心的读者还是会发现有一些细节需要完善,我们在使用 read_table 时,是需要指定

```
In [74]: df=pd.read_csv("G:\pybook\data2.txt")
         df
```

Out[74]:

		code	name	oDate	tDate
0	0	900949	东电B股	1997-09-22	2013-11-07
1	1	600253	天方药业	2000-12-27	2013-07-15
2	2	600991	广汽长丰	2004-06-14	2012-03-20
3	3	600263	路桥建设	2000-07-25	2012-03-01
4	4	600102	莱钢股份	1997-08-28	2012-02-28
5	5	600631	百联股份	1993-02-26	2011-08-23
6	6	600553	太行水泥	2002-08-22	2011-02-18
7	7	600003	ST东北高	1999-08-10	2010-02-26

图 11-5

```
In [82]: df=pd.read_table("G:\pybook\data2.csv")
         df
```

Out[82]:

	,code,name,oDate,tDate
0	0,900949,东电B股,1997-09-22,2013-11-07
1	1,600253,天方药业,2000-12-27,2013-07-15
2	2,600991,广汽长丰,2004-06-14,2012-03-20
3	3,600263,路桥建设,2000-07-25,2012-03-01
4	4,600102,莱钢股份,1997-08-28,2012-02-28
5	5,600631,百联股份,1993-02-26,2011-08-23
6	6,600553,太行水泥,2002-08-22,2011-02-18
7	7,600003,ST东北高,1999-08-10,2010-02-26

图 11-6

```
In [83]: df=pd.read_table("G:\pybook\data2.csv")
         df.head()
```

Out[83]:

	,code,name,oDate,tDate
0	0,900949,东电B股,1997-09-22,2013-11-07
1	1,600253,天方药业,2000-12-27,2013-07-15
2	2,600991,广汽长丰,2004-06-14,2012-03-20
3	3,600263,路桥建设,2000-07-25,2012-03-01
4	4,600102,莱钢股份,1997-08-28,2012-02-28

图 11-7

分隔符的。如图 11-8 所示。

4. 读取文本数据的重要方法

在处理很大的文件的时候，或找出大文件的参数集便于后续处理的时候，我们可以选择读取一部分。

比如上面的例子，我们选取前面 10 行进行读取，如图 11-9 所示。

```
In [84]: df=pd.read_table("G:\pybook\data2.csv",sep=",")
         df.head()
```

Out[84]:

		code	name	oDate	tDate
0	0	900949	东电B股	1997-09-22	2013-11-07
1	1	600253	天方药业	2000-12-27	2013-07-15
2	2	600991	广汽长丰	2004-06-14	2012-03-20
3	3	600263	路桥建设	2000-07-25	2012-03-01
4	4	600102	莱钢股份	1997-08-28	2012-02-28

图　11-8

```
In [85]: df=pd.read_table("G:\pybook\data2.csv",sep=",",nrows=10)
         df
```

Out[85]:

		code	name	oDate	tDate
0	0	900949	东电B股	1997-09-22	2013-11-07
1	1	600253	天方药业	2000-12-27	2013-07-15
2	2	600991	广汽长丰	2004-06-14	2012-03-20
3	3	600263	路桥建设	2000-07-25	2012-03-01
4	4	600102	莱钢股份	1997-08-28	2012-02-28
5	5	600631	百联股份	1993-02-26	2011-08-23
6	6	600553	太行水泥	2002-08-22	2011-02-18
7	7	600003	ST东北高	1999-08-10	2010-02-26
8	8	600842	中西药业	1994-03-11	2010-02-12
9	9	600607	上实医药	1992-03-27	2010-02-12

图　11-9

对于使用 Pandas 读取文本文件，我们做个总结，如表 11-1 所示。

表　11-1

函　数	说　　明
read_csv	从文件中读取带分隔符的数据。默认分隔符是逗号
read_table	从文件中读取带分隔符的数据。默认分隔符是制表符(\t)

11.2　数据规整化

数据分析和建模的大量编程工作都是用在数据准备上的：加载、清理、转换以及重塑。因为，有时候我们拿到的数据可能并不能满足我们数据处理应用的要求。

Python 和 Pandas 提供了一组高效、灵活的、高级的核心函数和算法，使我们能够轻松地将数据规整化为正确的形式。

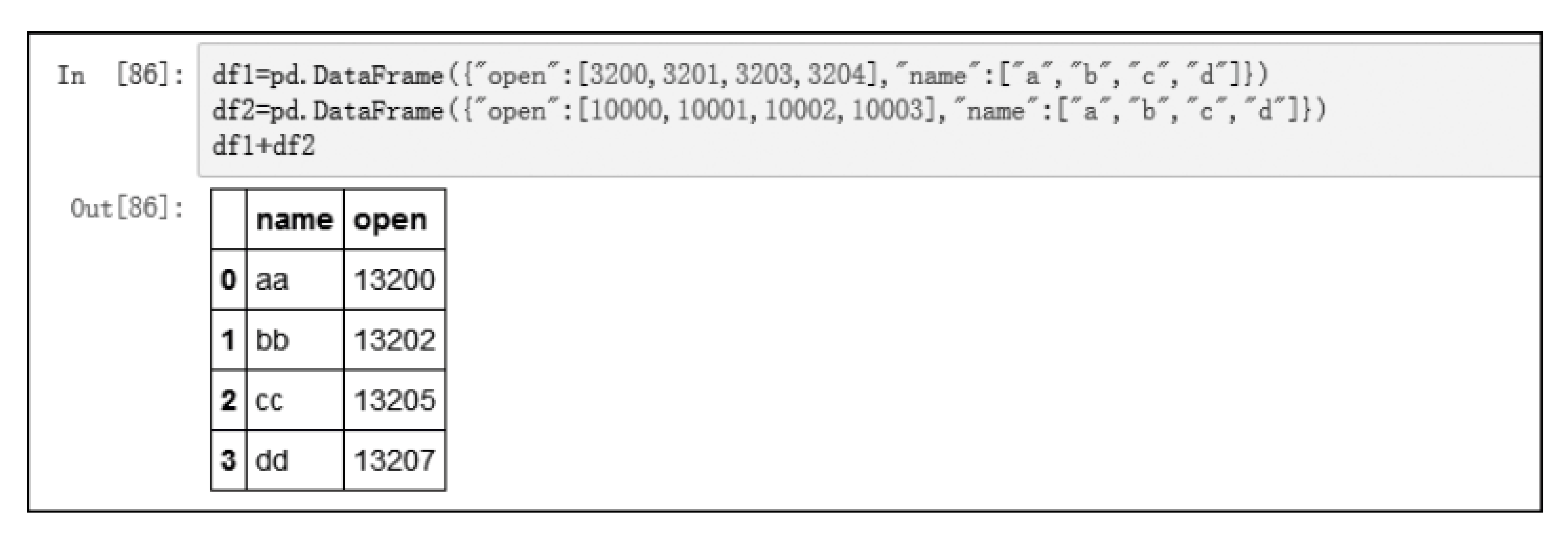
11.2.1　数据库风格的 DataFrame 合并

先介绍一下数据库中的两个概念：内连接和外连接。

（1）内连接需要的是两个表的某一行满足一定的条件(直接理解为：默认的是两个表的某一行相等的才返回这行的元素)。

（2）外连接返回的是两个表所有列标签的元素，该列标签没有元素，Pandas 中用 NaN 代替。

先举第一个例子，如图 11-10 所示。



```
In  [86]: df1=pd.DataFrame({"open":[3200,3201,3203,3204],"name":["a","b","c","d"]})
          df2=pd.DataFrame({"open":[10000,10001,10002,10003],"name":["a","b","c","d"]})
          df1+df2
Out[86]:
```

	name	open
0	aa	13200
1	bb	13202
2	cc	13205
3	dd	13207

图　11-10

如果我们直接用加法，会发现这个可能并不符合我们的预期。

第二个例子，如图 11-11 所示。

```
In  [87]: df1=pd.DataFrame({"open":[3200,3201,3203,3204],"name":["a","b","c","d"]})
          df2=pd.DataFrame({"open":[10000,10001,10002,10003],"date":["20170301","20170302"
                        ,"20170303","20170304"]})
          df1+df2
Out[87]:
```

	date	name	open
0	NaN	NaN	13200
1	NaN	NaN	13202
2	NaN	NaN	13205
3	NaN	NaN	13207

图　11-11

这个例子，可以看到，合并的时候如果列标签不一样，都是按照 NaN 处理的，当然我们也可以按一定规则进行数据补充。

第三个例子，如图 11-12 所示。

```
In  [88]: df1=pd.DataFrame({"open":[3200,3201,3203,3204],"name":["a","b","c","d"]})
          df2=pd.DataFrame({"open":[10000,10001,10002,10003],"name":["a","b","c","d"]})
          pd.merge(df1,df2)
Out[88]:
```

	name	open

图　11-12

我们知道有一个 merge 函数，但是结果如上。问题出在哪里呢？

第四个例子，如图 11-13 所示。

事实就是，在一般情况下，如果我们需要合并两个表格，那么我们采取的方法是设置 merge 的参数为“outer”。

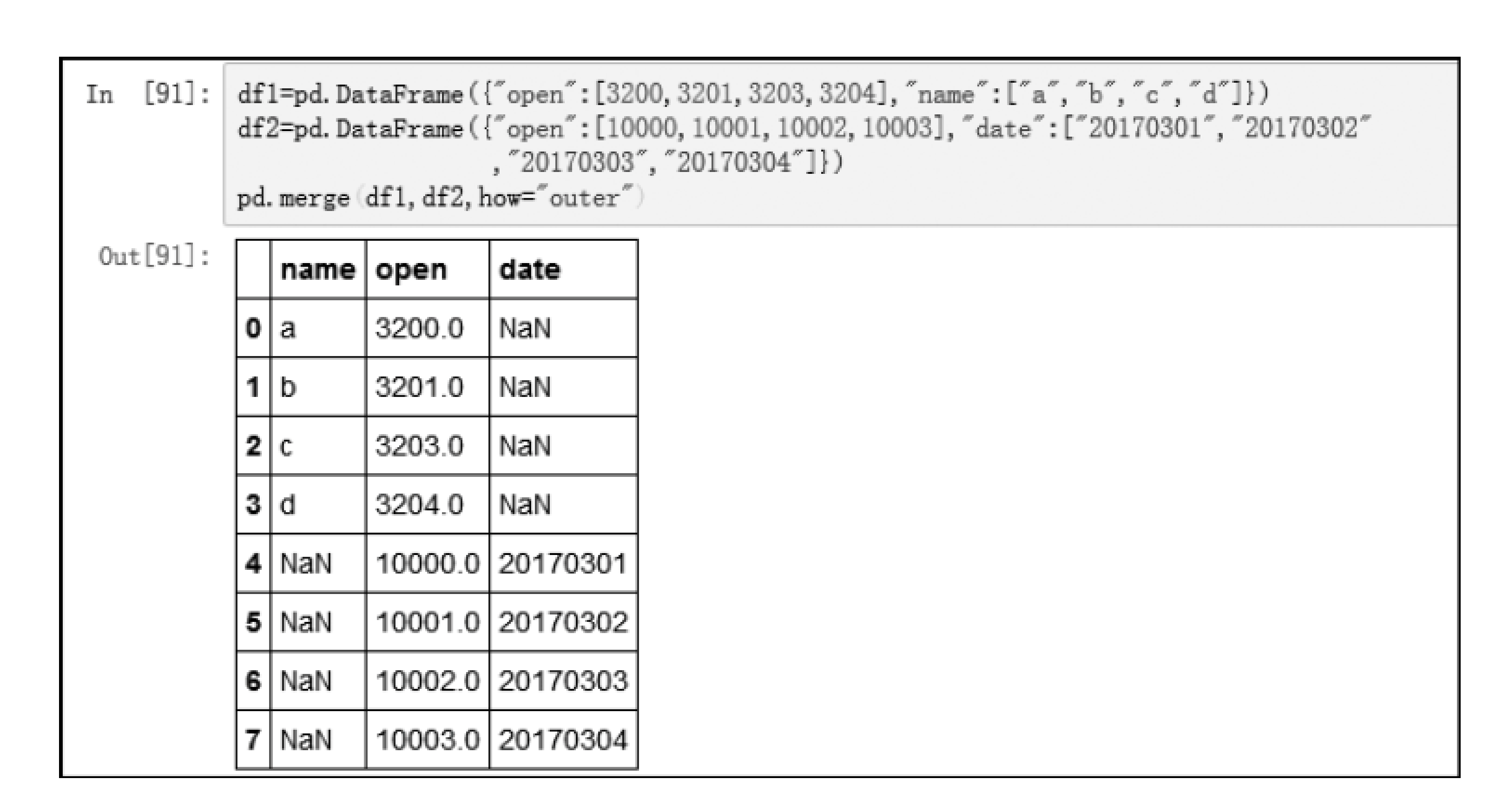
第五个例子，如图 11-14 所示。

```
In  [89]: df1=pd.DataFrame({"open":[3200,3201,3203,3204],"name":["a","b","c","d"]})
          df2=pd.DataFrame({"open":[10000,10001,10002,10003],"name":["a","b","c","d"]})
          pd.merge(df1,df2,how="outer")
```

Out[89]:

	name	open
0	a	3200.0
1	b	3201.0
2	c	3203.0
3	d	3204.0
4	a	10000.0
5	b	10001.0
6	c	10002.0
7	d	10003.0

图 11-13



```
In  [91]: df1=pd.DataFrame({"open":[3200,3201,3203,3204],"name":["a","b","c","d"]})
          df2=pd.DataFrame({"open":[10000,10001,10002,10003],"date":["20170301","20170302"
                      ,"20170303","20170304"]})
          pd.merge(df1,df2,how="outer")
```

Out[91]:

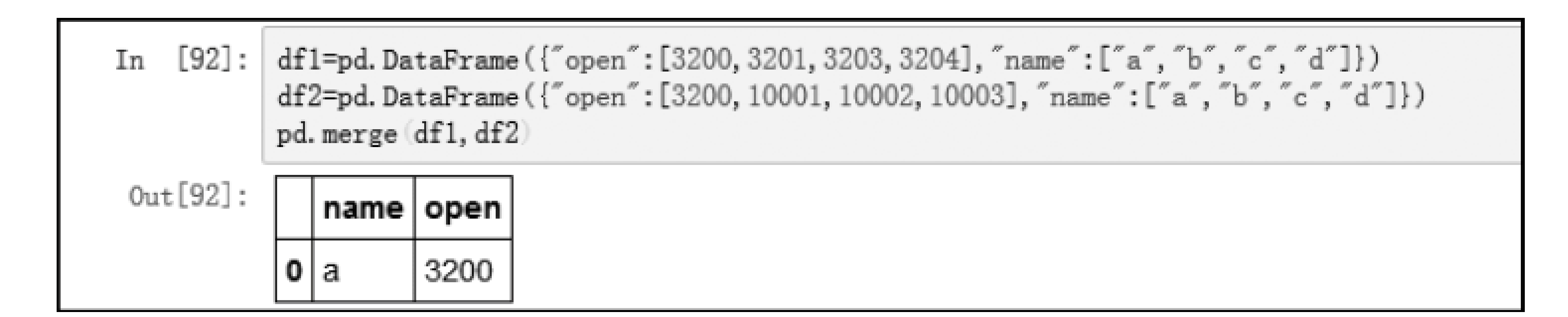
	name	open	date
0	a	3200.0	NaN
1	b	3201.0	NaN
2	c	3203.0	NaN
3	d	3204.0	NaN
4	NaN	10000.0	20170301
5	NaN	10001.0	20170302
6	NaN	10002.0	20170303
7	NaN	10003.0	20170304

图 11-14

通过参数设置“outer”，我们发现通过使用这样的方法，得到了我们想要的结果。

在默认情况下，merge 的参数是“inner”。

第六个例子，如图 11-15 所示。



```
In  [92]: df1=pd.DataFrame({"open":[3200,3201,3203,3204],"name":["a","b","c","d"]})
          df2=pd.DataFrame({"open":[3200,10001,10002,10003],"name":["a","b","c","d"]})
          pd.merge(df1,df2)
```

Out[92]:

	name	open
0	a	3200

图 11-15

这个例子进一步验证，merge 的默认方法使用的是内连接。

如果读者明白了内连接和外连接，则数据库还有左连接和右连接就比较容易理解了，这里就不介绍了，有兴趣的读者可自行百度。

11.2.2 轴向连接

大家看到的，合并默认的也是针对行的，那么针对列怎么操作呢？

第一个例子，如图 11-16 所示。

```
In [93]: data1=pd.Series([11,22],index=["high","low"])
         data2=pd.Series([33,44,55],index=["ma5","ma20","ma60"])
         data3=pd.Series([10000,20000],index=["volume","code"])
         pd.concat([data1,data2,data3])

Out[93]: high         11
         low          22
         ma5          33
         ma20         44
         ma60         55
         volume    10000
         code      20000
         dtype: int64
```

图 11-16

可以看到的，默认的方法是按照行的标签进行合并的。

第二个例子，如图 11-17 所示。

```
In [94]: data1=pd.Series([11,22],index=["high","low"])
         data2=pd.Series([33,44,55],index=["ma5","ma20","ma60"])
         data3=pd.Series([10000,20000],index=["volume","code"])
         pd.concat([data1,data2,data3],axis=1)

Out[94]:
```

	0	1	2
code	NaN	NaN	20000.0
high	11.0	NaN	NaN
low	22.0	NaN	NaN
ma20	NaN	44.0	NaN
ma5	NaN	33.0	NaN
ma60	NaN	55.0	NaN
volume	NaN	NaN	10000.0

图 11-17

如果使用 axis=1，则我们会发现产生了一个 DataFrame。

11.2.3 移除重复数据

举例如图 11-18 所示。

```
In [95]: df=pd.DataFrame({"open":[3200,3200,3203,3204],"name":["a","a","c","d"]})
         df.drop_duplicates()

Out[95]:
```

	name	open
0	a	3200
2	c	3203
3	d	3204

图 11-18

Drop_duplicates 方法用于返回一个移除了重复行的 DataFrame。

11.2.4 利用函数进行数据转换

在对数据进行处理的时候,我们可能希望方便地处理得到一些真正想要的数据,比如全部转换成为期望为 0 的数据等。

第一个例子,如图 11-19 所示。

```
In  [96]: data=pd.Series([1,2,3,4,5],index=["high","low","open","close","ma5"])
          data.map(lambda x:x**2)

Out[96]: high      1
         low       4
         open      9
         close    16
         ma5      25
         dtype: int64
```

图 11-19

上面的方法就是把原来的 Series 的每个元素都平方并返回。

第二个例子,如图 11-20 所示。

```
In  [97]: df=pd.DataFrame({"open":[3200,3201,3203,3204],"name":["a","b","c","d"]})
          df["open"].map(lambda x:x+100000)

Out[97]: 0    103200
         1    103201
         2    103203
         3    103204
         Name: open, dtype: int64
```

图 11-20

这个例子中,我们针对 DataFrame 对象中的 open 列每个元素都加 10 000 并返回。

11.2.5 替换数据

如果我们把 map 方法看作是针对全部或局部的数据替换,那么我们可以把 replace 看作是针对特定的数据替换。

前面我们学习的 fillna 方法也可以看作是数据替换的一种特殊情况。

我们看第一个例子,如图 11-21 所示。

```
In  [98]: data=pd.Series([1,2,3,4,5],index=["high","low","open","close","ma5"])
          data.replace(3,100000)

Out[98]: high          1
         low           2
         open     100000
         close         4
         ma5           5
         dtype: int64
```

图 11-21

第二个例子,如图 11-22 所示。

```
In  [99]: pd.merge(df1,df2)
          data.replace({3:100000,4:99999})

Out[99]: high          1
         low           2
         open     100000
         close     99999
         ma5           5
         dtype: int64
```

图 11-22

11.2.6 Groupby 操作

先举个例子。

比如,我们本来有一组数据,如图 11-23 所示。

```
In  [117]: df=pd.DataFrame({"open":[1,2,3,4,5],"high":[6,7,8,9,10],"low":[10,20,30,40,50],"close":[60,70,80,90,100]},
                 index=["20170301","20170302","20170303","20170304","20170305"])
           df["ma5"]=["Q1","Q1","Q1","Q2","Q2",]
           df

Out[117]:
```

	close	high	low	open	ma5
20170301	60	6	10	1	Q1
20170302	70	7	20	2	Q1
20170303	80	8	30	3	Q1
20170304	90	9	40	4	Q2
20170305	100	10	50	5	Q2

图 11-23

现在我们需要计算前三天和后两天的各个开高低收价格的平均值。我们可以先添加一列,表示我们需要计算的变量,如图 11-24 所示。

```
In  [118]: grouped = df.groupby(df['ma5'])
           grouped.mean()

Out[118]:
```

	close	high	low	open
ma5				
Q1	70.0	7.0	20.0	2.0
Q2	95.0	9.5	45.0	4.5

图 11-24

这里我们使用的是 grouby 函数。按 ma5 进行分组,并计算 ma5 列的平均值,我们可以访问 df,并根据 ma5 调用 groupby。

当然,我们可以仅仅针对某一列进行这样的计算。如图 11-25 所示。

```
In  [119]: grouped = df["open"].groupby(df['ma5'])
           grouped.mean()

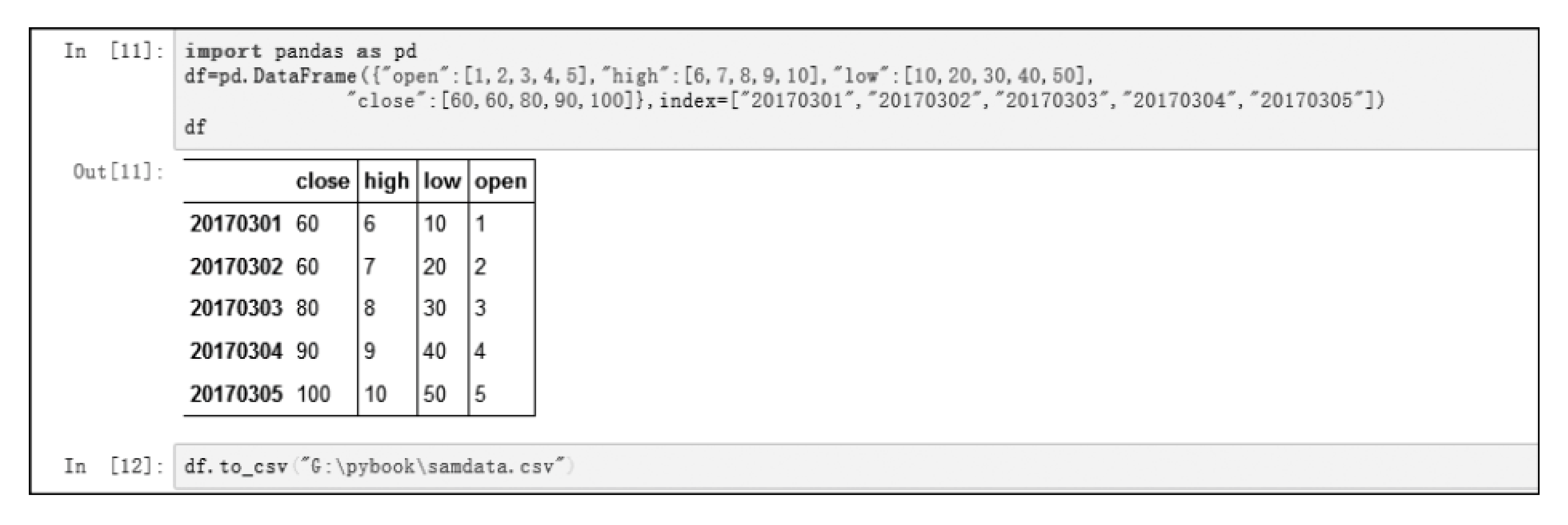
Out[119]: ma5
          Q1    2.0
          Q2    4.5
          Name: open, dtype: float64
```

图 11-25

注意：变量 grouped 是一个 Groupby 对象，它实际上还没有进行任何计算，只是含有一些有关分组键 df['ma5']的中间数据而已，然后我们可以调用 Groupby 的 mean 方法来计算分组平均值。

11.3 保存数据

这里我们只是介绍保存为 csv 格式的，其他的格式基本上也是类似的。如图 11-26 所示。



```
In [11]: import pandas as pd
         df=pd.DataFrame({"open":[1,2,3,4,5],"high":[6,7,8,9,10],"low":[10,20,30,40,50],
                        "close":[60,60,80,90,100]},index=["20170301","20170302","20170303","20170304","20170305"])
         df
```

Out[11]:

	close	high	low	open
20170301	60	6	10	1
20170302	60	7	20	2
20170303	80	8	30	3
20170304	90	9	40	4
20170305	100	10	50	5

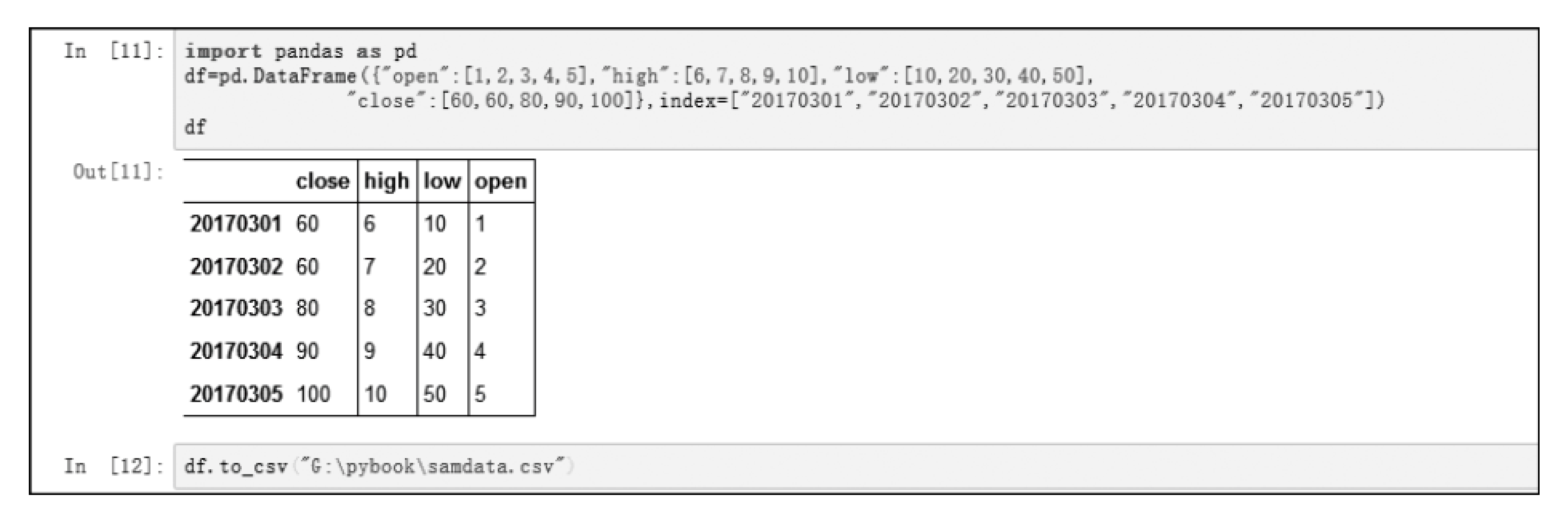
```
In [12]: df.to_csv("G:\pybook\samdata.csv")
```

图 11-26

注意：变量 grouped 是一个 Groupby 对象，它实际上还没有进行任何计算，只是含有一些有关分组键 df['ma5']的中间数据而已，然后我们可以调用 Groupby 的 mean 方法来计算分组平均值。

11.3 保存数据

这里我们只是介绍保存为 csv 格式的，其他的格式基本上也是类似的。如图 11-26 所示。



```
In [11]: import pandas as pd
         df=pd.DataFrame({"open":[1,2,3,4,5],"high":[6,7,8,9,10],"low":[10,20,30,40,50],
                 "close":[60,60,80,90,100]},index=["20170301","20170302","20170303","20170304","20170305"])
         df
Out[11]:
```

	close	high	low	open
20170301	60	6	10	1
20170302	60	7	20	2
20170303	80	8	30	3
20170304	90	9	40	4
20170305	100	10	50	5

```
In [12]: df.to_csv("G:\pybook\samdata.csv")
```

图 11-26

第 12 章 绘图和可视化

在使用 Numpy 进行学习统计计算时是枯燥的，大量的数据令我们很头疼，所以我们需要把它图形化显示。

Matplotlib 是一个 Python 的图形框架，类似于 MATLAB 和 R 语言。

12.1 使用 matplotlib

通常的引入约定是：import matplotlib. pyplot as plt 。先看一个简单的例子，如图 12-1 所示。

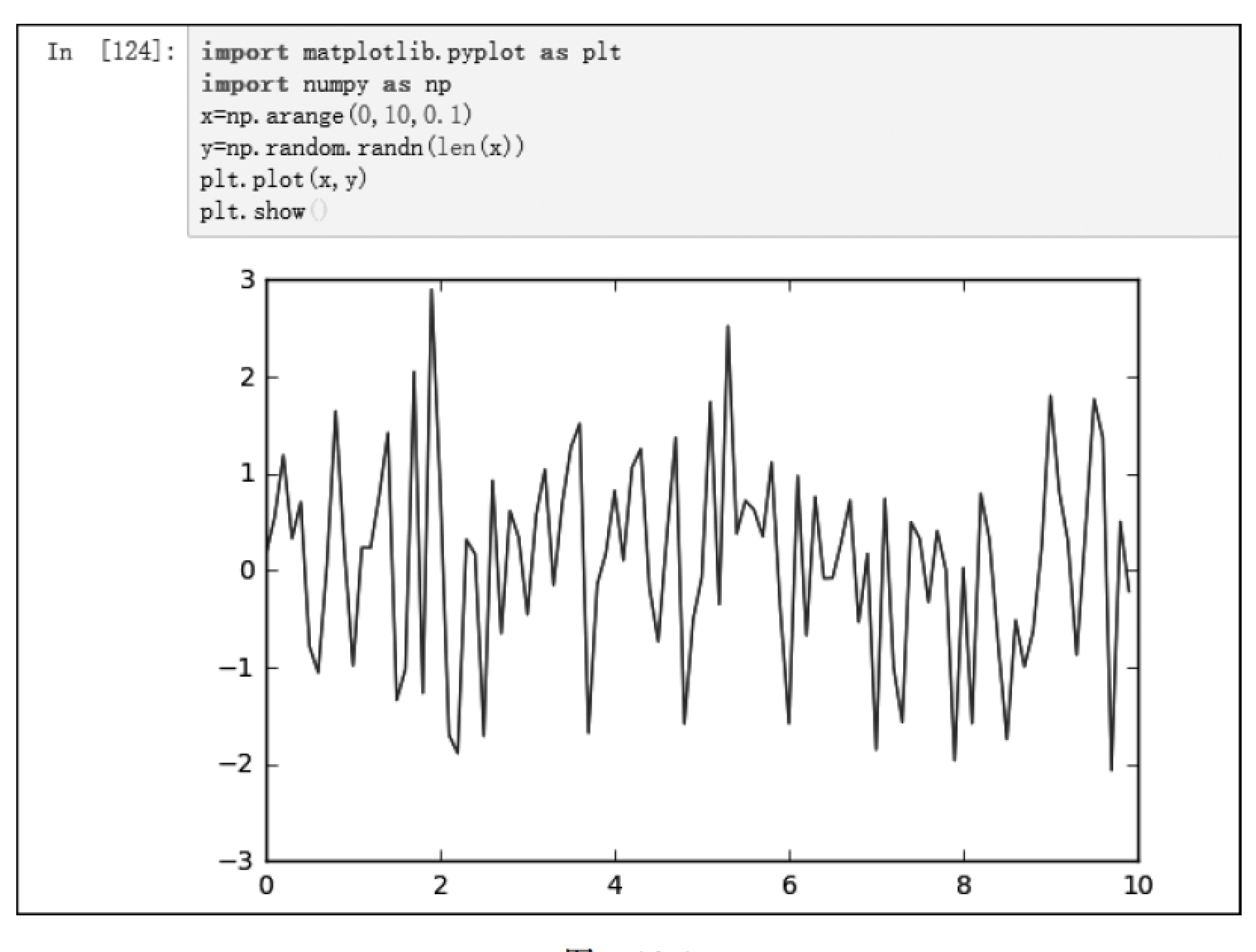


图 12-1

绘图大致分为以下几步。

第一步，导入 matplotlib. pyplot。

第二步，设置自变量和因变量。

第三步，调用 plot 函数绘图，以及 show 函数显示图像。

当然，这样的图像很粗糙，我们需要设置标题、颜色等属性，接下来我们逐步进行详细介绍。

1. Figure 和 Subplot

Matplotlib 的图像都位于 figure 对象中。我们可以用 plt. figure 创建一个新的 Figure。这个就相当于提供了一块画布,我们可以在上面画一张图或几张图。当然我们要这样操作的话,就不能直接画图,必须先指定创建一个或多个图片窗口(subplot)。

比如:

```
ax1=fig.add_subplot(2,2,1)
```

这个代码的意思是:图像是 2×2 的,就是说有 2 行 2 列,4 个图的,而且当前选中的是第一个。如果我们把后面的三个图都创建出来的话,则

```
ax2=fig.add_subplot(2,2,2)
ax3=fig.add_subplot(2,2,3)
ax4=fig.add_subplot(2,2,4)
```

如果这个时候,我们画图的话,默认的情况下是针对最后一个图片窗口画图的。比如:

如果我们把 4 个图都画上,可直接调用每个图片窗口的 plot 方法,如图 12-2 所示。

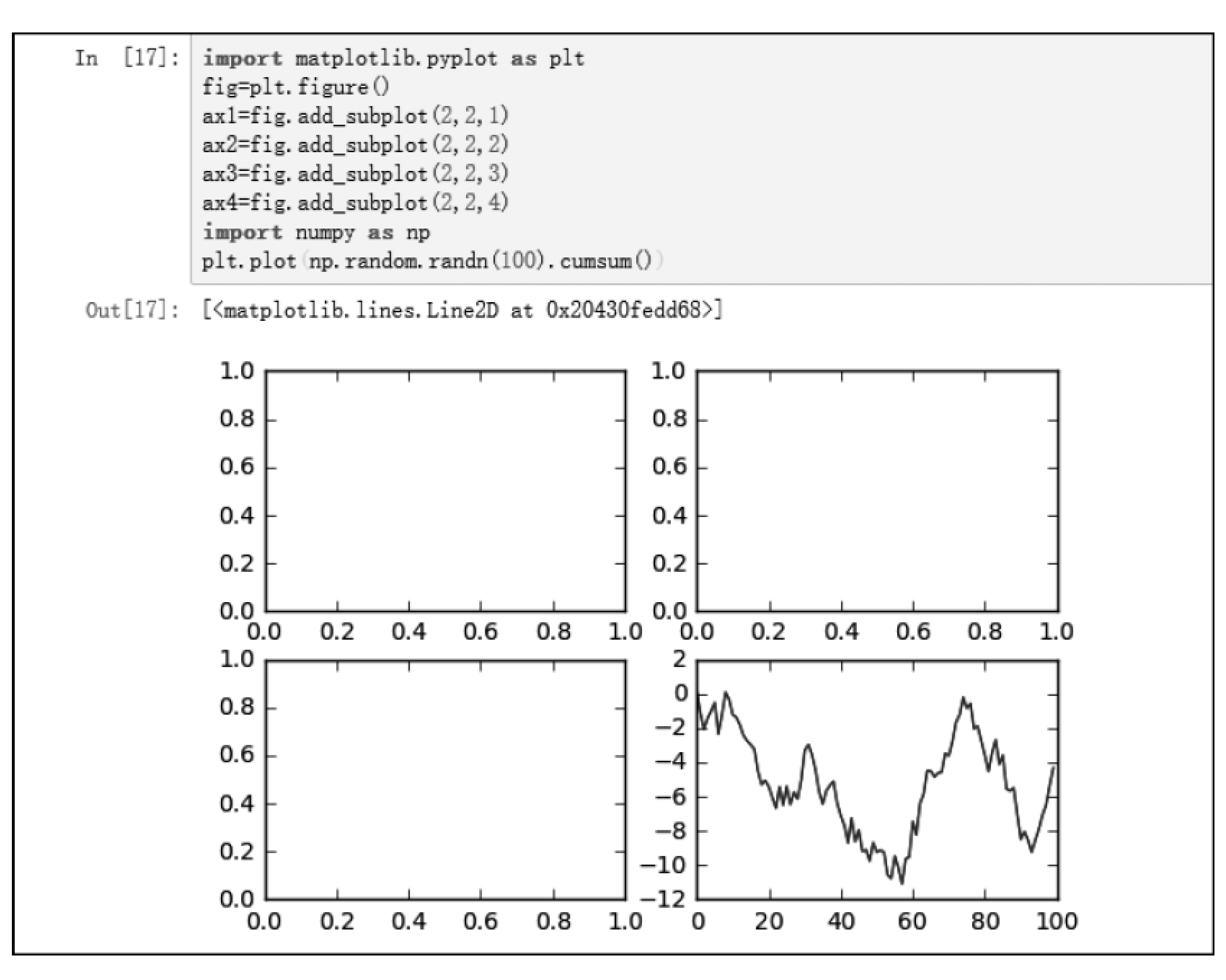


图 12-2

2. 颜色、标记和线型

Matplotlib 的 plot 函数接受一组 *X* 和 *Y* 坐标,还可以接受一个表示颜色和线型的字符串缩写。例如:我们在图 12-3 第四个图中用虚线进行绘制。

当然,我们可以更规范地使用如下形式:

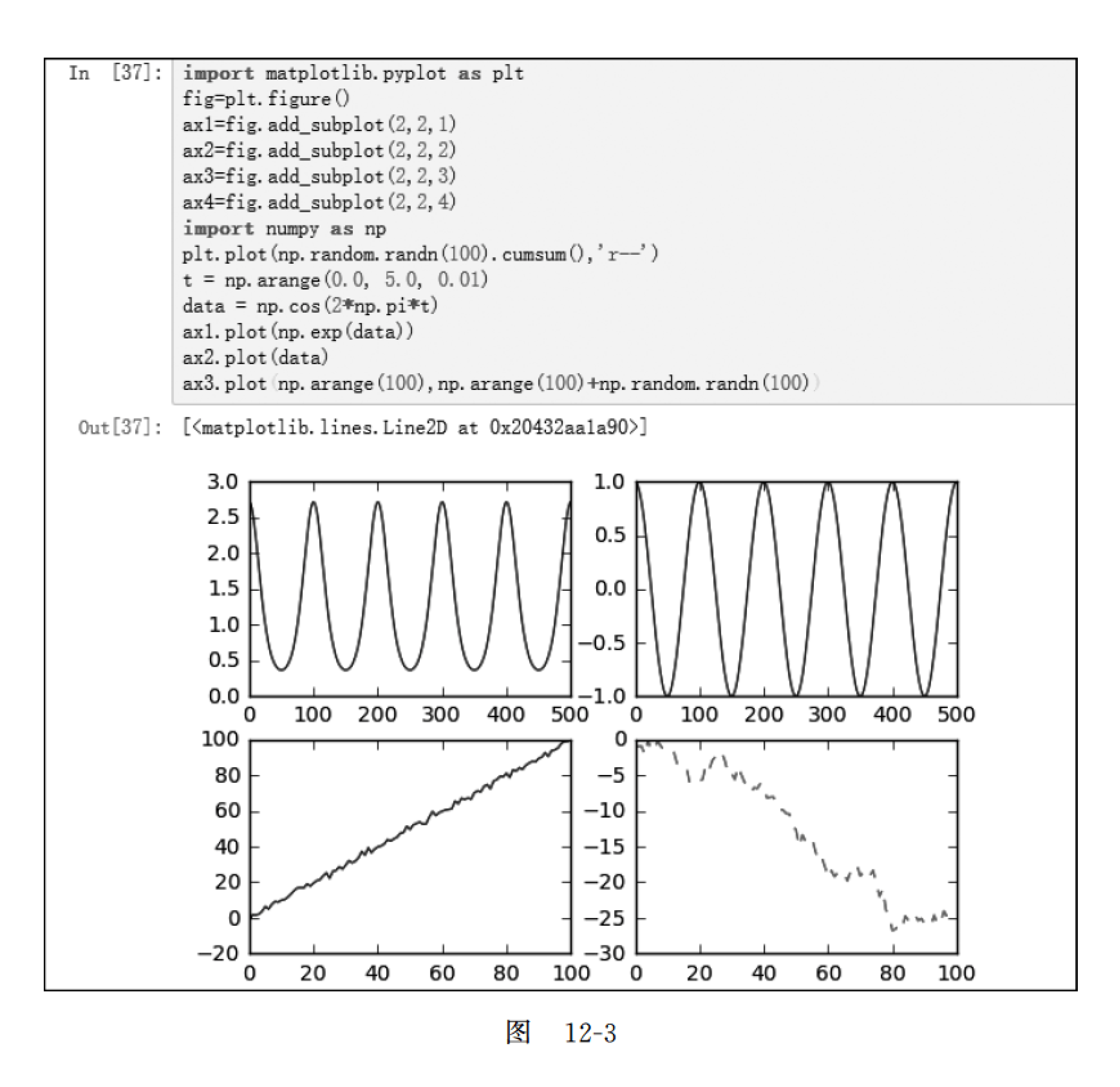


图 12-3

plot(x,y,linestyle="—",color="r"),读者希望使用各种颜色可以通过查询官方文档或者百度。

线性图还可以加上一些标记(marker),以强调实际的数据点,如图 12-4 所示。

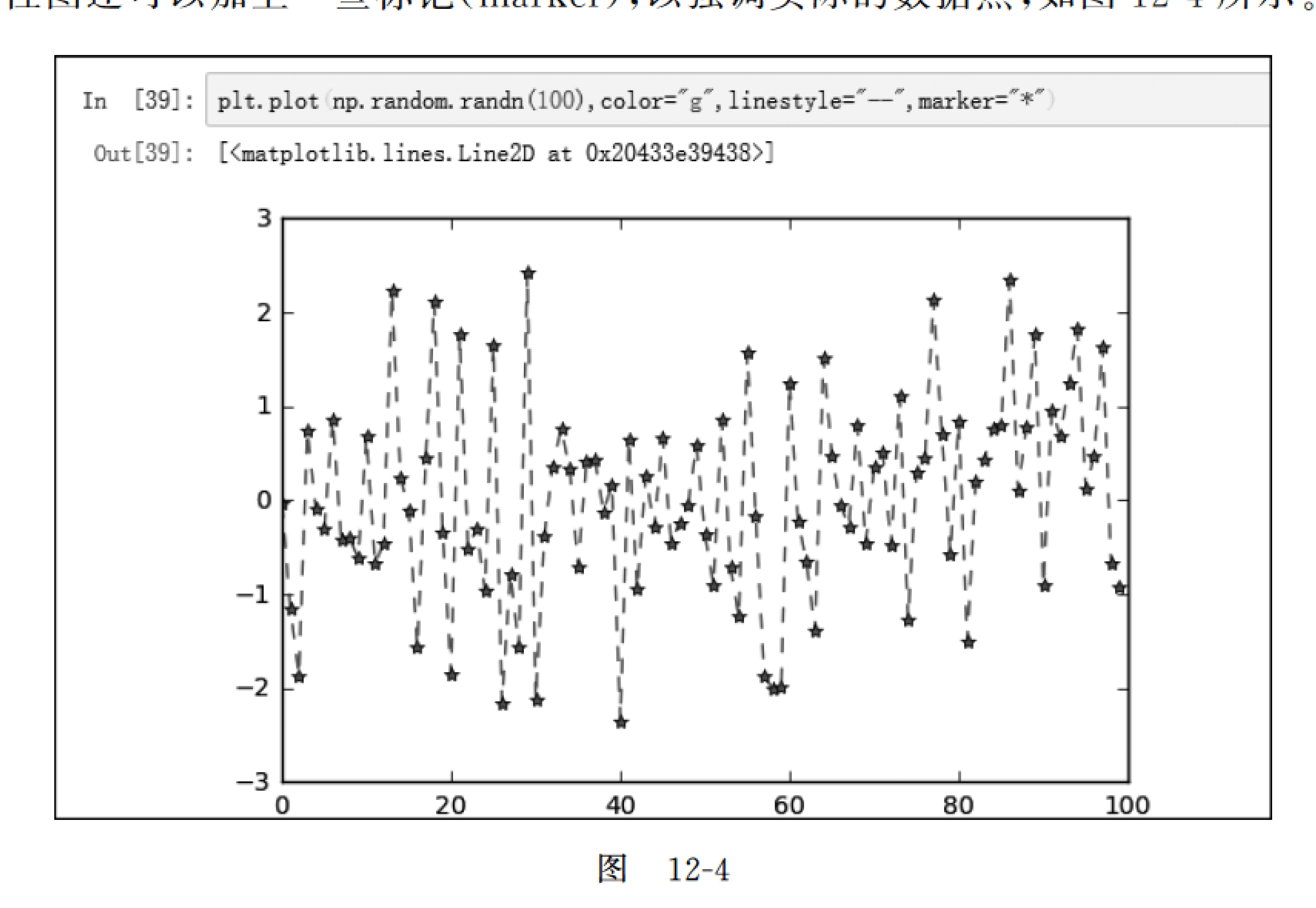


图 12-4

3. 刻度、标签和图例

我们仍然通过下面的例子来说明操作方法，如图 12-5 所示。

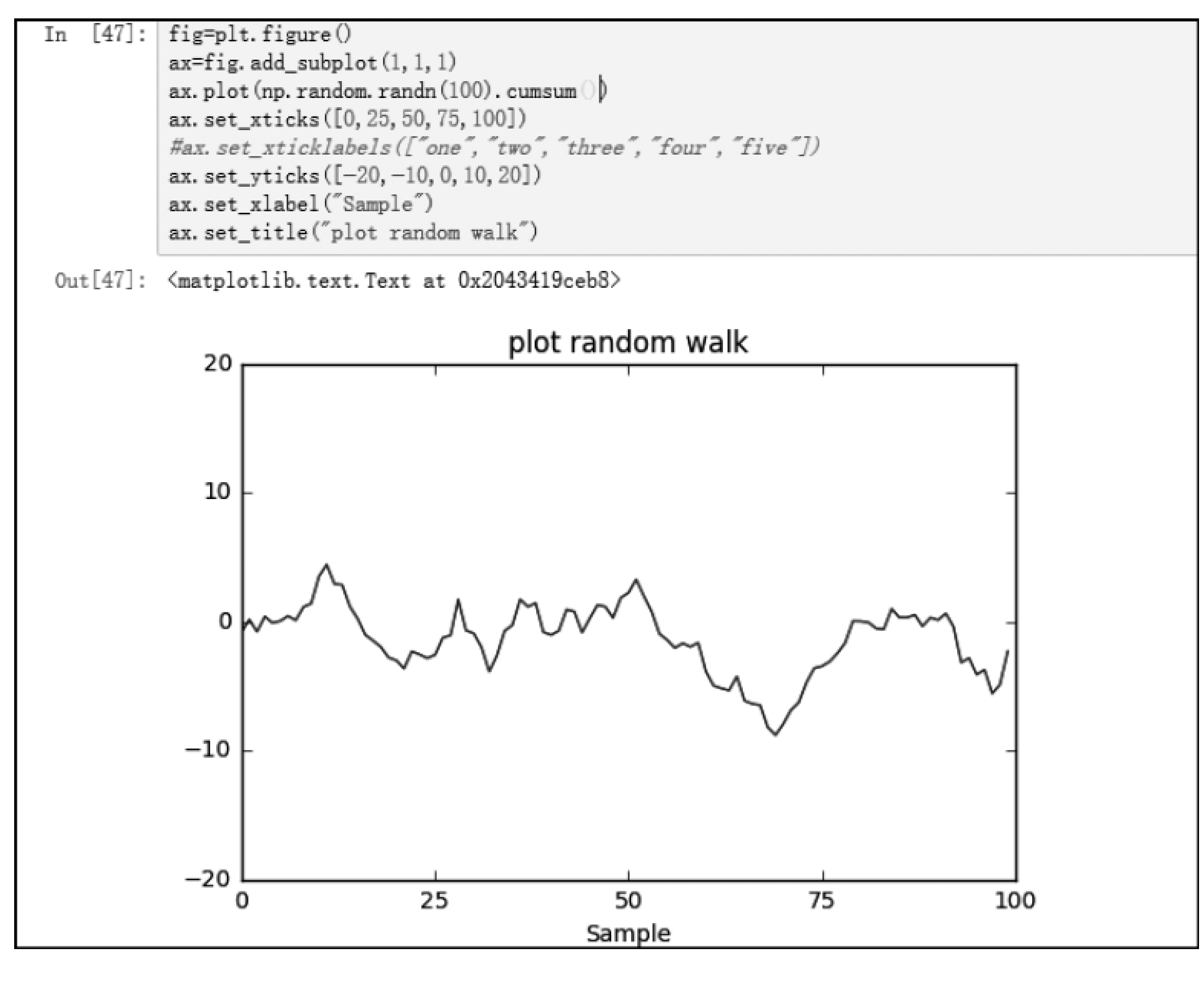


图 12-5

4. 一个图片窗口中画多幅图并添加标签

如果我们需要就多组数据进行绘图，并在同一个图片窗口中进行比较，我们就需要做个标签，如图 12-6 所示。

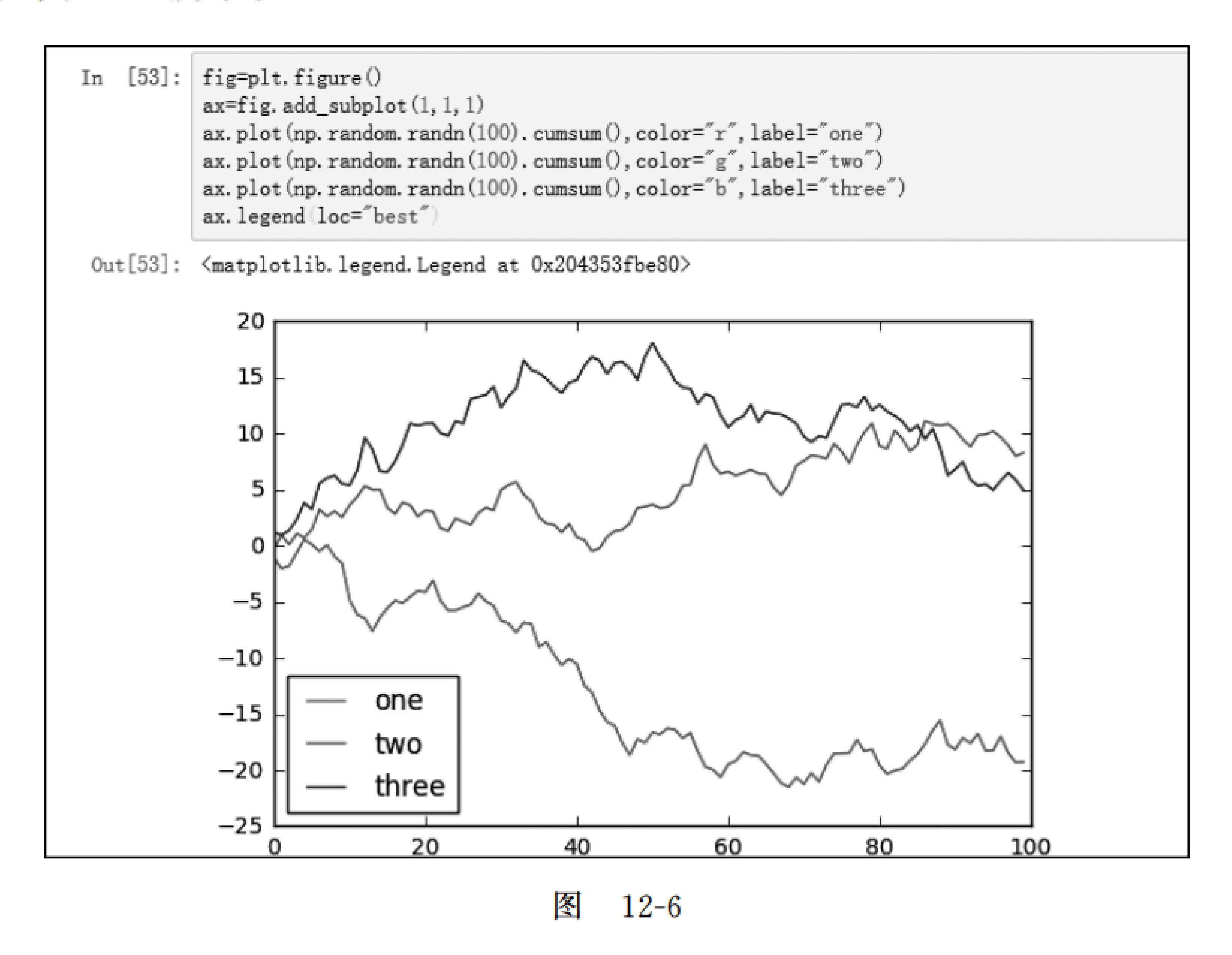


图 12-6

在一个图片中画多个图片是直接不断地调用 plot 方法，在添加图例的时候，我们可以通过 label 属性进行设置。如上图所示，loc 告诉 matplotlib 将图放在哪个位置，一般情况下推荐使用“best”，因为系统会自动选择最适合的位置。

5. 保存图表到文件

利用 plt. savefig 可以保存当前的图表到文件。

如图 12-7 所示，我们把图片保存到 G 盘 pybook 文件夹里，那么我们查看该文件夹时可以看到如图 12-8 所示。

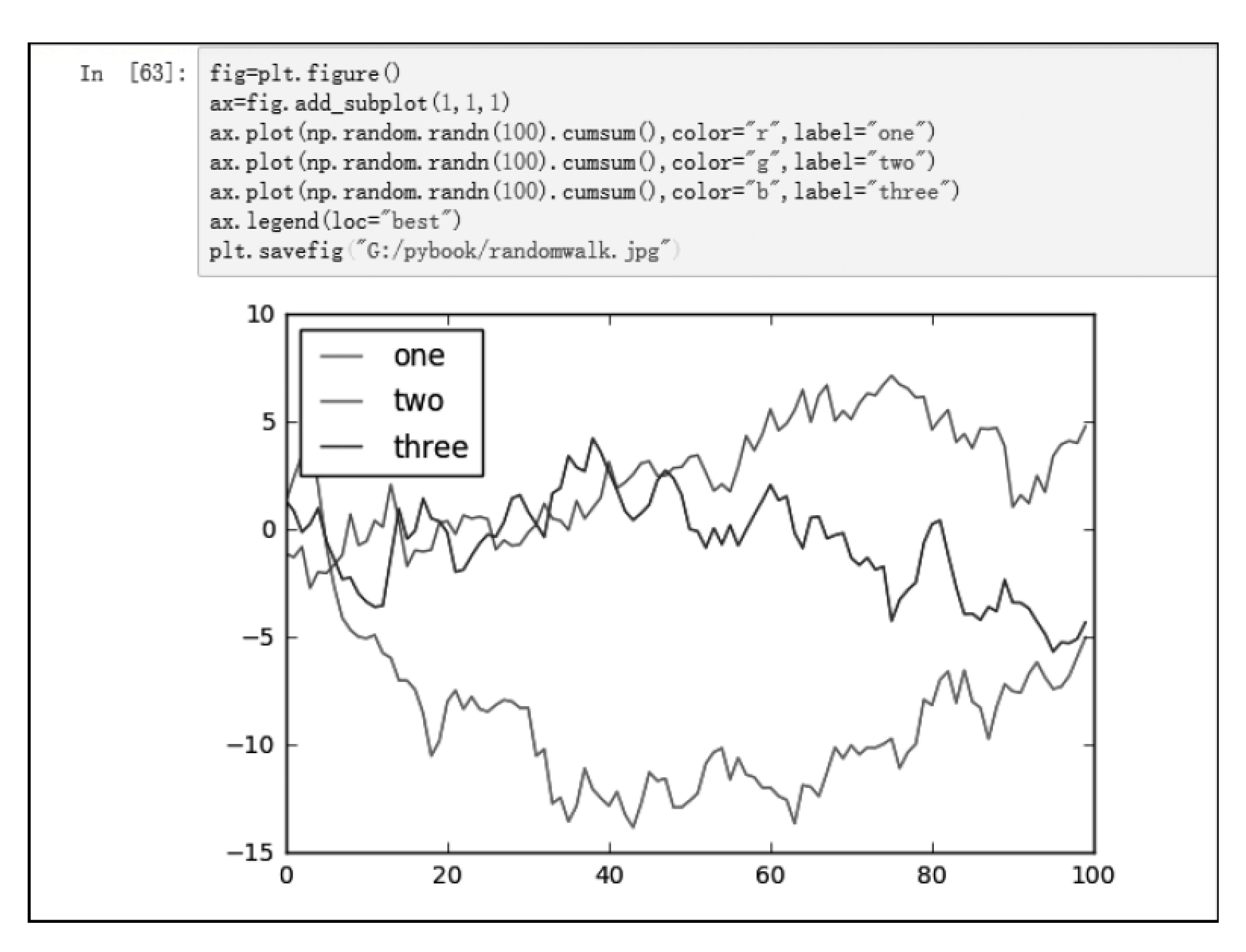


图 12-7

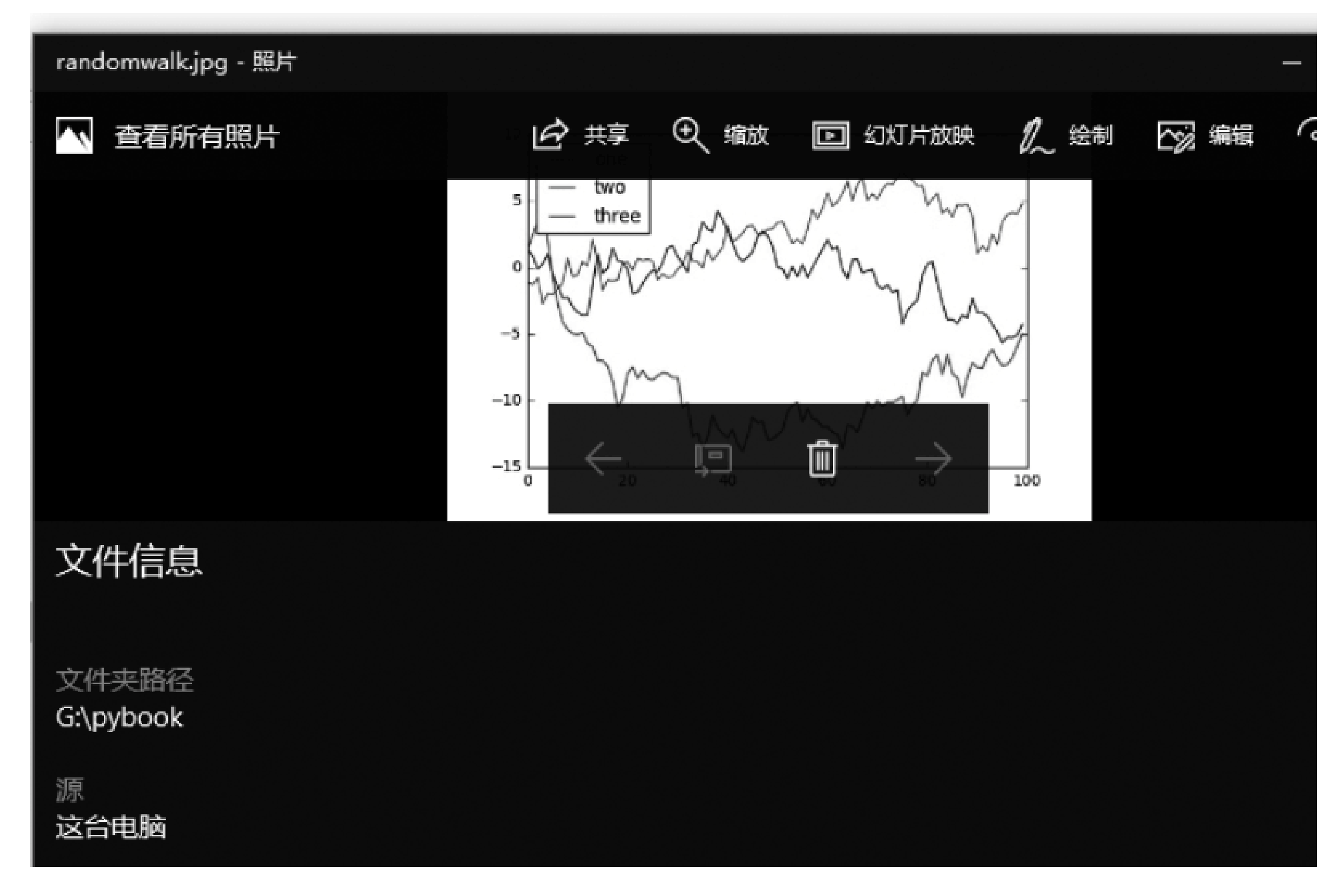


图 12-8

12.2　Pandas 中的绘图函数

Pandas 提供了更简单的一些绘图方法。

1. 针对 Series 绘图

举例,如图 12-9 所示。

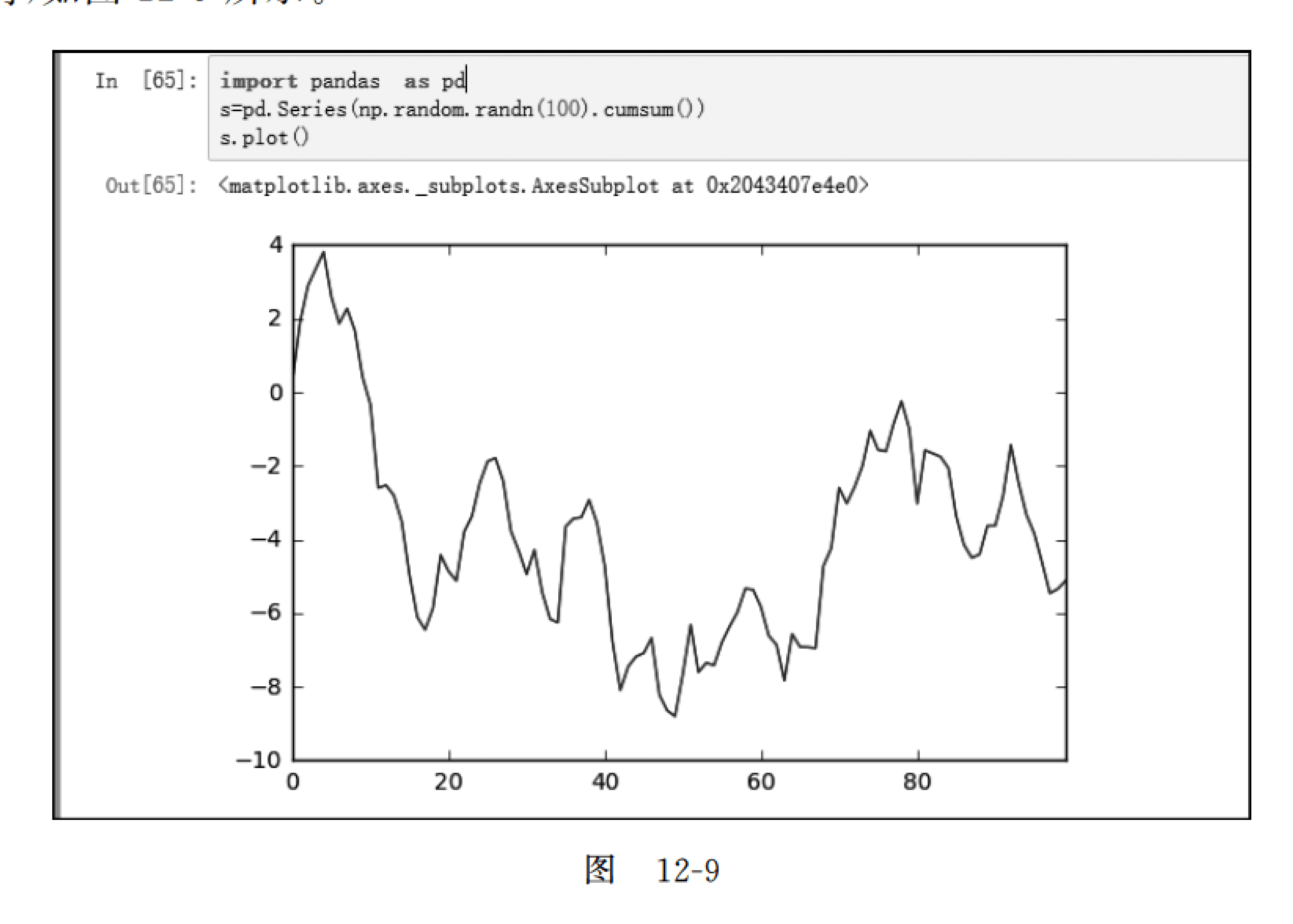


图　12-9

也就是说,用 Pandas 中的 plot 画图很简单,直接调用即可。

2. 针对 DataFrame 绘图

对于 DataFrame 中的各列数据分别在同一个图片窗口中进行画图,如图 12-10 所示。

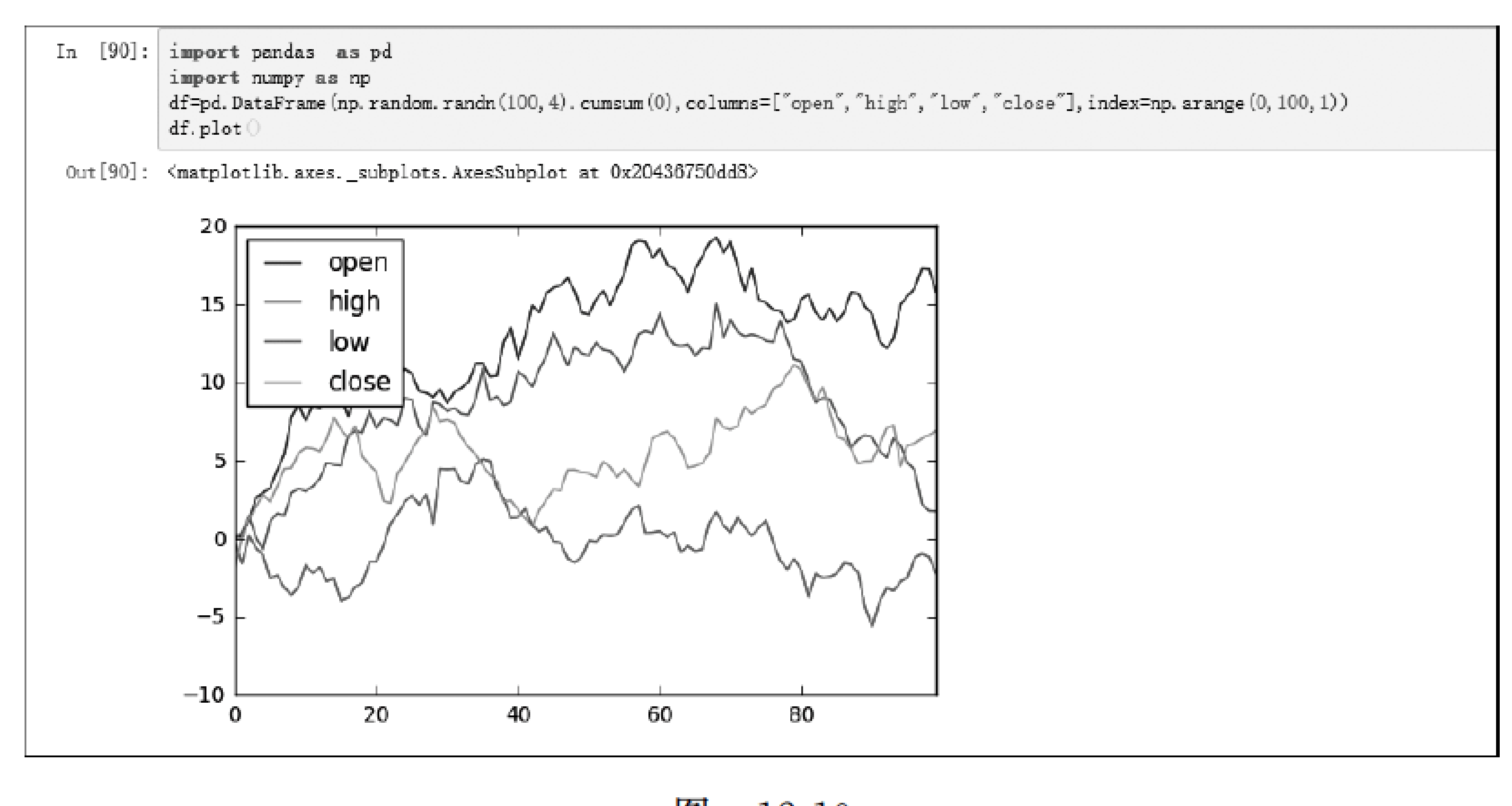


图　12-10

关于利用 Pandas 更详细的画图方法,读者可以自行研究。这里只是希望看到直接调用 Pandas 是最方便的,特别是在做金融数据分析时应尽可能使用 Series 和 DataFrame。

第 13 章 金融数据分析

13.1 金融时间序列

在金融数据分析中，时间序列是用得最多的了，毕竟大部分人还是拿交易数据在进行分析。所以时间序列是一种重要的结构化数据格式。而且很多时间序列是固定频率的，比如我们一般都是分析日线数据，或者 30 分钟数据，等等。时间序列数据的意义取决于具体的应用场景，主要包括以下几个：

时间戳(timestamp)：特定的时刻。

固定时期(period)：如 2017 年 3 月或 2016 年。

时间间隔(interval)：由起始时间戳和结束时间戳表示。时期可以看作是时间间隔的特例。

1. 日期和时间类型以及工具

这里我们主要介绍 datatime 库，如图 13-1 所示。

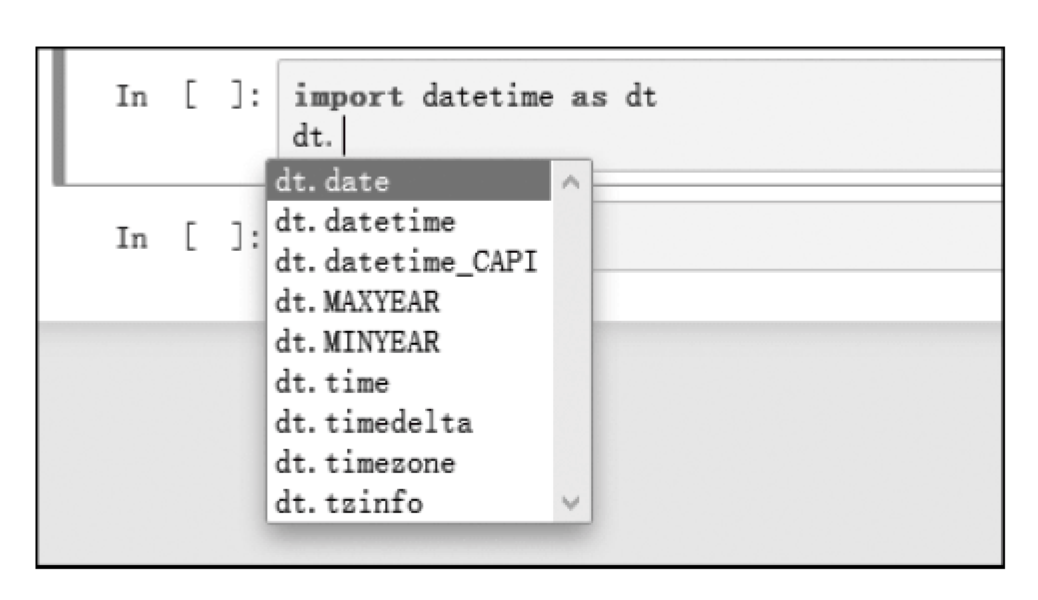


图 13-1

在这个库中，我们用得最多的是 datetime 这个对象。这里有两个 datetime，前者是一个库，后者是一个对象。如图 13-2 所示。

```
In [21]: import datetime as dt
         now=dt.datetime.now()
         now

Out[21]: datetime.datetime(2017, 3, 27, 8, 30, 10, 140590)

In [22]: import datetime as dt
         now=dt.datetime.now()
         print(now.year,now.month,now.day)

         2017 3 27
```

图 13-2

Datetime.timedelta 表示两个 datetime 对象之间的时间差。

```
In [49]: import datetime as dt
         delta=dt.datetime(2017,3,27,0,0)-dt.datetime(2016,1,1,0,0)
         delta

Out[49]: datetime.timedelta(451)

In [50]: delta.days

Out[50]: 451
```

图 13-3

这里是按照天数进行计算的。

2. 检查带有重复索引的时间序列

有时候，我们抓取数据或者拿到的数据可能需要去掉重复的，比如某一天的数据重复了，我们就需要去掉。举例如图 13-4 所示：

```
In [58]: import pandas as pd
         import numpy as np
         dates=pd.DatetimeIndex(["3/1/2017","3/2/2017","3/2/2017","3/3/2017","3/3/2017"])
         data=pd.Series(np.arange(5),index=dates)
         data

Out[58]: 2017-03-01    0
         2017-03-02    1
         2017-03-02    2
         2017-03-03    3
         2017-03-03    4
         dtype: int32

In [57]: data.index.is_unique

Out[57]: False

In [59]: data["3/2/2017"]

Out[59]: 2017-03-02    1
         2017-03-02    2
         dtype: int32
```

图 13-4

假设我们需要对非唯一时间戳的数据进行聚合。一个很好的办法就是使用 groupby，并传入 level＝0（索引的唯一一层），如图 13-5 所示。

```
In [84]: grouped=data.groupby(level=0)
         grouped.unique()

Out[84]: 2017-03-01       [0]
         2017-03-02    [1, 2]
         2017-03-03    [3, 4]
         dtype: object

In [85]: grouped.mean()

Out[85]: 2017-03-01    0.0
         2017-03-02    1.5
         2017-03-03    3.5
         dtype: float64
```

图 13-5

特别地，如果是高频数据，比如国内股指期货交易规则是每 500 毫秒会发送一次成交数据，但是可能因为网速等原因，会遇到一些重复数据，那么这个时候可能就需要根据具

体情况做相应的去重处理了。

3. 重采样以及频率转换

重采样(resample)指的是将时间序列从一个频率转换成另一个频率的处理过程。很多时候,我们遇到的就是把高频率的数据转换成低频率的数据,专业术语也叫降采样。比如我们可能获得的数据是1分钟数据,为了交易或者策略的需要,我们需转换成5分钟、1小时或日数据等。

比如,我们先生成一个日线数据,如图13-6所示。

```
In [92]: import pandas as pd
         import numpy as np
         date1=pd.date_range("1/1/2016",periods=100,freq="D")
         date1
         #ts=pd.Series(np.random.randn(len(date1)),index=date1)
         #ts.resample("M").mean()

Out[92]: DatetimeIndex(['2016-01-01', '2016-01-02', '2016-01-03', '2016-01-04',
                        '2016-01-05', '2016-01-06', '2016-01-07', '2016-01-08',
                        '2016-01-09', '2016-01-10', '2016-01-11', '2016-01-12',
                        '2016-01-13', '2016-01-14', '2016-01-15', '2016-01-16',
                        '2016-01-17', '2016-01-18', '2016-01-19', '2016-01-20',
                        '2016-01-21', '2016-01-22', '2016-01-23', '2016-01-24',
                        '2016-01-25', '2016-01-26', '2016-01-27', '2016-01-28',
                        '2016-01-29', '2016-01-30', '2016-01-31', '2016-02-01',
                        '2016-02-02', '2016-02-03', '2016-02-04', '2016-02-05',
                        '2016-02-06', '2016-02-07', '2016-02-08', '2016-02-09',
                        '2016-02-10', '2016-02-11', '2016-02-12', '2016-02-13',
                        '2016-02-14', '2016-02-15', '2016-02-16', '2016-02-17',
                        '2016-02-18', '2016-02-19', '2016-02-20', '2016-02-21',
                        '2016-02-22', '2016-02-23', '2016-02-24', '2016-02-25',
                        '2016-02-26', '2016-02-27', '2016-02-28', '2016-02-29',
                        '2016-03-01', '2016-03-02', '2016-03-03', '2016-03-04',
                        '2016-03-05', '2016-03-06', '2016-03-07', '2016-03-08',
```

图　13-6

接下来,如果我们要转换成月线数据的话,进行如下操作,如图13-7所示。

```
In [93]: import pandas as pd
         import numpy as np
         date1=pd.date_range("1/1/2016",periods=100,freq="D")
         #date1
         ts=pd.Series(np.random.randn(len(date1)),index=date1)
         ts.resample("M").mean()

Out[93]: 2016-01-31   -0.166621
         2016-02-29   -0.125994
         2016-03-31    0.087146
         2016-04-30   -0.195669
         Freq: M, dtype: float64
```

图　13-7

图13-7就是转换后的月线数据,且采用的是取均值的办法。记住这里是采样后的数据也是一个对象,这个对象也有mean、max等方法,但没有how这个参数的设置。

4. OHLC 采样

在金融领域中,开高低收(开盘价、最高价、最低价、收盘价)是常见的数据形式,如图13-8所示。

当然因为这里无法查看最高价和最低价,这里的最高价和最低价就是根据开盘价和收盘价比较而得出的。

5. 通过 groupby 进行采样

groupby 函数可能是最强大的采样工具了,如图13-9所示。

```
In  [101]: import pandas as pd
           import numpy as np
           date2=pd.date_range("1/1/2016",periods=100,freq="T")
           ts=pd.Series(np.arange(100),index=date2)
           ts.resample("5min").ohlc()
```

Out[101]:

	open	high	low	close
2016-01-01 00:00:00	0	4	0	4
2016-01-01 00:05:00	5	9	5	9
2016-01-01 00:10:00	10	14	10	14
2016-01-01 00:15:00	15	19	15	19
2016-01-01 00:20:00	20	24	20	24

图 13-8

```
In  [103]: date3=pd.date_range("1/1/2016",periods=100,freq="D")
           ts=pd.Series(np.arange(100),index=date3)
           ts.groupby(lambda x:x.month).mean()

Out[103]: 1    15
          2    45
          3    75
          4    95
          dtype: int32
```

图 13-9

建议读者进一步熟练使用 groupby 这个函数。

13.2 TuShare 介绍

TuShare 是一个免费、开源的 Python 财经数据接口包，官网是：http://tushare.org/。主要实现对股票等金融数据从数据采集、清洗加工到数据存储的过程，能够为金融分析人员提供快速、整洁和多样的便于分析的数据，为他们在数据获取方面极大地减轻工作量，使他们更加专注于策略和模型的研究与实现上。考虑到 Python Pandas 包在金融量化分析中体现出的优势，TuShare 返回的绝大部分的数据格式都是 Pandas DataFrame 类型，非常便于利用 Pandas/NumPy/Matplotlib 进行数据分析和可视化。

TuShare 的数据主要来源于网络，读者可以关注"挖地兔"的微信公众号，该作者会定期发布 TuShare 的最新动态及有价值的金融数据分析与处理方面的教程和文章。

TuShare 的数据包括交易数据、投资参考数据、股票分类数据、基本面数据、宏观经济数据、新闻事件数据、龙虎榜数据、银行间同业拆放利率等。

关于安装 TuShare，读者可以看前面介绍的方法，自行安装，并可登录 TuShare 的官方网站，上面也有详细介绍。

接下来，我们按照数据分析的流程做一个大盘指数的图形以及包含 5 日、20 日、60 日的均线图。

第一步：获取数据。如图 13-10 所示。

一般来讲，数据过多，可能我们很难直观地看到数据的一些基本情况，比如有多少行、

```
In [110]: import tushare as ts
          data=ts.get_hist_data("sh")
          data
```

Out[110]:

	open	high	close	low	volume	price_change	p_change	ma
date								
2017-03-24	3247.350	3275.210	3269.450	3241.120	2197779.25	20.900	0.64	325
2017-03-23	3245.810	3262.090	3248.550	3221.930	1930291.50	3.330	0.10	324
2017-03-22	3246.220	3255.780	3245.220	3229.130	1897316.50	-16.390	-0.50	325
2017-03-21	3250.250	3262.220	3261.610	3246.700	1627193.00	10.800	0.33	325
2017-03-20	3241.110	3251.130	3250.810	3228.120	1705484.25	13.360	0.41	324
2017-03-17	3271.870	3274.190	3237.450	3232.280	2005832.25	-31.490	-0.96	324
2017-03-16	3247.160	3269.770	3268.940	3247.160	1894160.00	27.180	0.84	323
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

图 13-10

多少列以及哪些列。我们可以用下面的方法予以统计，如图 13-11 所示。

```
In [123]: data.columns

Out[123]: Index(['open', 'high', 'close', 'low', 'volume', 'price_change', 'p_change',
                 'ma5', 'ma10', 'ma20', 'v_ma5', 'v_ma10', 'v_ma20'],
                dtype='object')

In [124]: data.index

Out[124]: Index(['2017-03-24', '2017-03-23', '2017-03-22', '2017-03-21', '2017-03-20',
                 '2017-03-17', '2017-03-16', '2017-03-15', '2017-03-14', '2017-03-13',
                 ...
                 '2014-04-09', '2014-04-08', '2014-04-04', '2014-04-03', '2014-04-02',
                 '2014-04-01', '2014-03-31', '2014-03-28', '2014-03-27', '2014-03-26'],
                dtype='object', name='date', length=733)
```

图 13-11

比如，我们看到数据共有 733 行数据，包括从 2014 年 3 月 26 号到 2017 年 3 月 24 日的数据。

第二步：保存数据，如图 13-12 所示。

```
In [156]: import tushare as ts
          data=ts.get_hist_data("sh")
          data.to_csv("G:\pybook\index.csv")
```

图 13-12

第三步，再打开数据，如图 13-13 所示。

```
In [157]: import pandas as pd
          df=pd.read_csv("G:\pybook\index.csv",parse_dates=True,index_col=0)
```

图 13-13

这里需要注意的是关于 read_csv 函数是个非常重要的函数，但是它的参数又是特别多的，我们没必要都去记住，只是需要掌握几个重要的就可以了。这里有两个参数是很重要的，大家必须掌握。

Parse_dates 主要是为了使得原来获得数据中的日期形式转换成为 Datetime Index 形式的，这样才能使用前面我们提及的 resample 函数以及 groupby 函数，这一点非常重要，否则，使用采样函数的时候是会出错的，如图 13-14 所示。

```
In [158]: import tushare as ts
          data=ts.get_hist_data("sh")
          #data.to_csv("G:\pybook\index.csv")
          data.groupby(lambda x:x.month).mean()

          ---------------------------------------------------------------------------
          AttributeError                            Traceback (most recent call last)
          <ipython-input-158-e1ca22d45ebb> in <module>()
                2 data=ts.get_hist_data("sh")
                3 #data.to_csv("G:\pybook\index.csv")
          ----> 4 data.groupby(lambda x:x.month).mean()

          C:\Program Files\Anaconda3\lib\site-packages\pandas\core\generic.py in groupby(self, by, axis, level, as_
          *kwargs)
             3776         return groupby(self, by=by, axis=axis, level=level, as_index=as_index,
             3777                        sort=sort, group_keys=group_keys, squeeze=squeeze,
          -> 3778                        **kwargs)
             3779
             3780     def asfreq(self, freq, method=None, how=None, normalize=False):
```

图　13-14

设定 index_col＝0 或者 false 的意思是 pandas 不适用第一列作为行索引。

下面我们查看是否有时间重复的数据，图 13-15。

```
In [126]: data.index.is_unique

Out[126]: True
```

图　13-15

如果我们希望数据按照时间先后进行排序，那么我们最好使用 sort_index 进行重新排序，如图 13-16 所示。

```
In [177]: data2=data.sort_index(ascending=True)
          data2
```

Out[177]:

	open	high	close	low	volume	price_change
date						
2014-03-26	2070.574	2074.572	2063.670	2057.648	1026111.19	-3.641
2014-03-27	2060.812	2073.982	2046.588	2042.713	1191493.75	-17.082
2014-03-28	2046.851	2060.134	2041.712	2035.243	1216819.62	-4.876
2014-03-31	2043.045	2048.134	2033.306	2024.185	943565.38	-8.406
2014-04-01	2031.005	2050.681	2047.460	2028.096	832865.75	14.154
2014-04-02	2049.423	2060.778	2058.988	2046.742	1026585.25	11.528
2014-04-03	2063.497	2066.007	2043.702	2037.447	1086184.25	-15.286
2014-04-04	2037.552	2060.104	2058.831	2035.221	831890.56	15.129
2014-04-08	2054.530	2102.452	2098.284	2052.900	1333709.88	39.453

图　13-16

第四步：我们通过日线数据使用 groupby 函数获得月线数据，如图 13-17 所示。

第五步：绘图，如图 13-18 所示。

In [161]:
```
import pandas as pd
df=pd.read_csv("G:\pybook\index.csv",parse_dates=True,index_col=0)
df.groupby(lambda x:x.month).mean()
```

Out[161]:

	open	high	close	low	volume	price_change	p_change	ma5
1	3139.821293	3179.890655	3137.909155	3098.813172	2.380120e+06	-13.282138	-0.392586	3162.909569
2	3066.661633	3091.426469	3070.521000	3042.312714	2.046312e+06	2.711837	0.089592	3064.485592
3	3124.256433	3150.893552	3135.402149	3103.831507	2.695414e+06	11.153000	0.346567	3114.695552
4	3080.443274	3109.829000	3089.908581	3054.145629	2.988997e+06	10.019484	0.240484	3068.287661
5	3112.835311	3142.992934	3114.892836	3077.451410	2.461427e+06	2.633328	0.074098	3109.103377
6	3273.519508	3305.620230	3272.171672	3216.950295	2.891445e+06	-5.119902	-0.087869	3286.246787
7	2976.001254	3023.155030	2982.799597	2936.594612	3.012867e+06	-6.127239	-0.073582	2991.691672
8	2952.659954	2985.478738	2957.084523	2921.662785	2.464501e+06	-5.168492	-0.114154	2970.525246
9	2812.090279	2839.282246	2818.813557	2790.441295	2.024488e+06	-1.431639	-0.009344	2819.788803
10	2904.680059	2928.520804	2913.367451	2885.192314	2.339803e+06	9.448588	0.315490	2894.960000

图 13-17

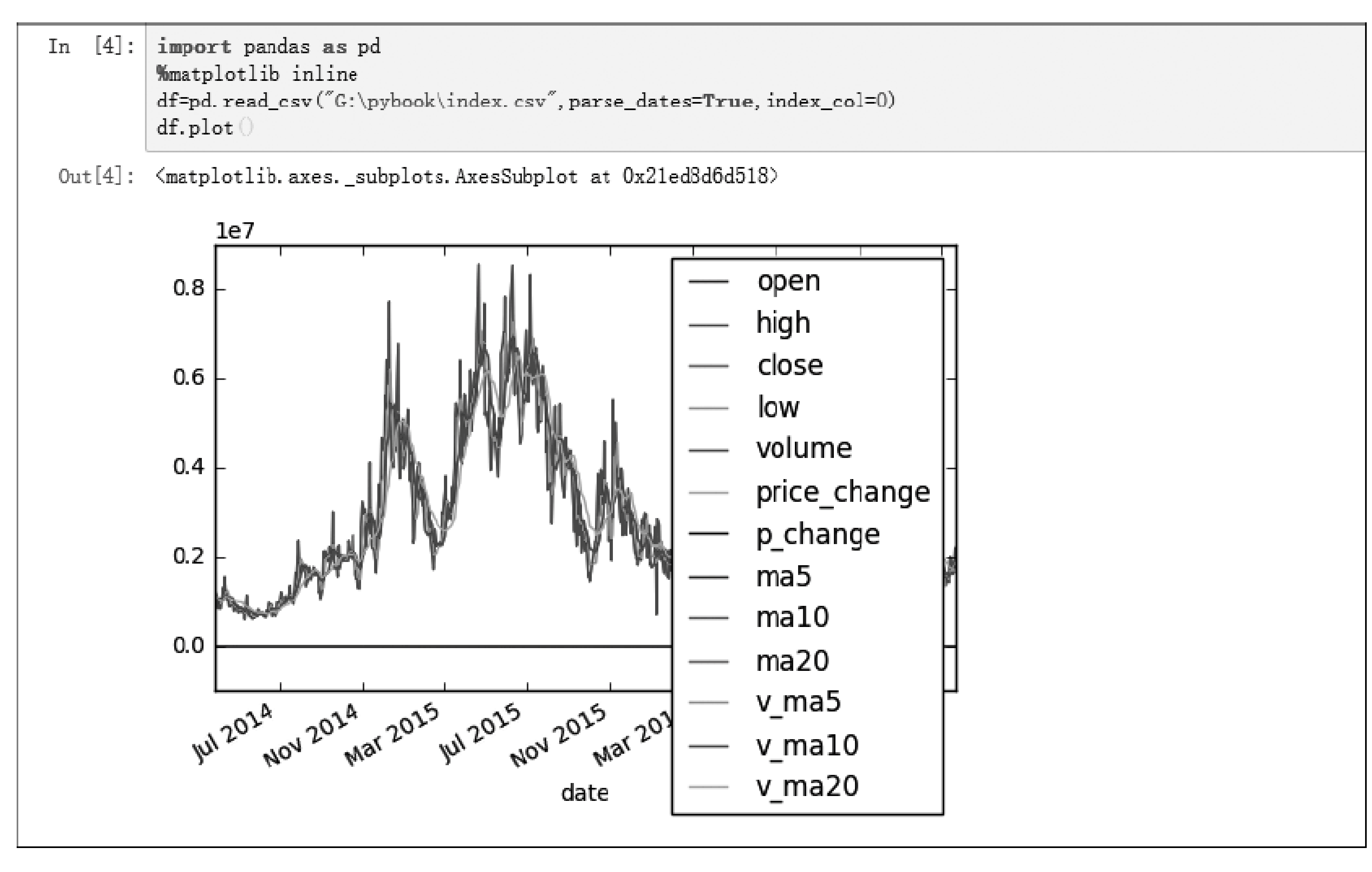


图 13-18

当然这样全部画出来,看起来很凌乱,我们可以选取一部分进行画图。

比如,我们在图 13-19 中画出了收盘数据、5 日均线、20 日均线。如果我们还想画出 60 日均线的话,就可以使用 rolling 函数了,如图 13-20 所示。

这里需要注意的几个事项如下:

(1) df=df.sort_index(ascending=True)。

这个语句说明的是:因为涉及自己计算均值,所以我们先统一把数据按照日期升序进行排列。

(2) ma60=ma60.fillna(df.close[59])。

这个语句说明的是:60 日均线前 60 个数据都是 NaN,所以我们把 df.close[59]这个数据,也就是第 60 个数据替换为 NaN。

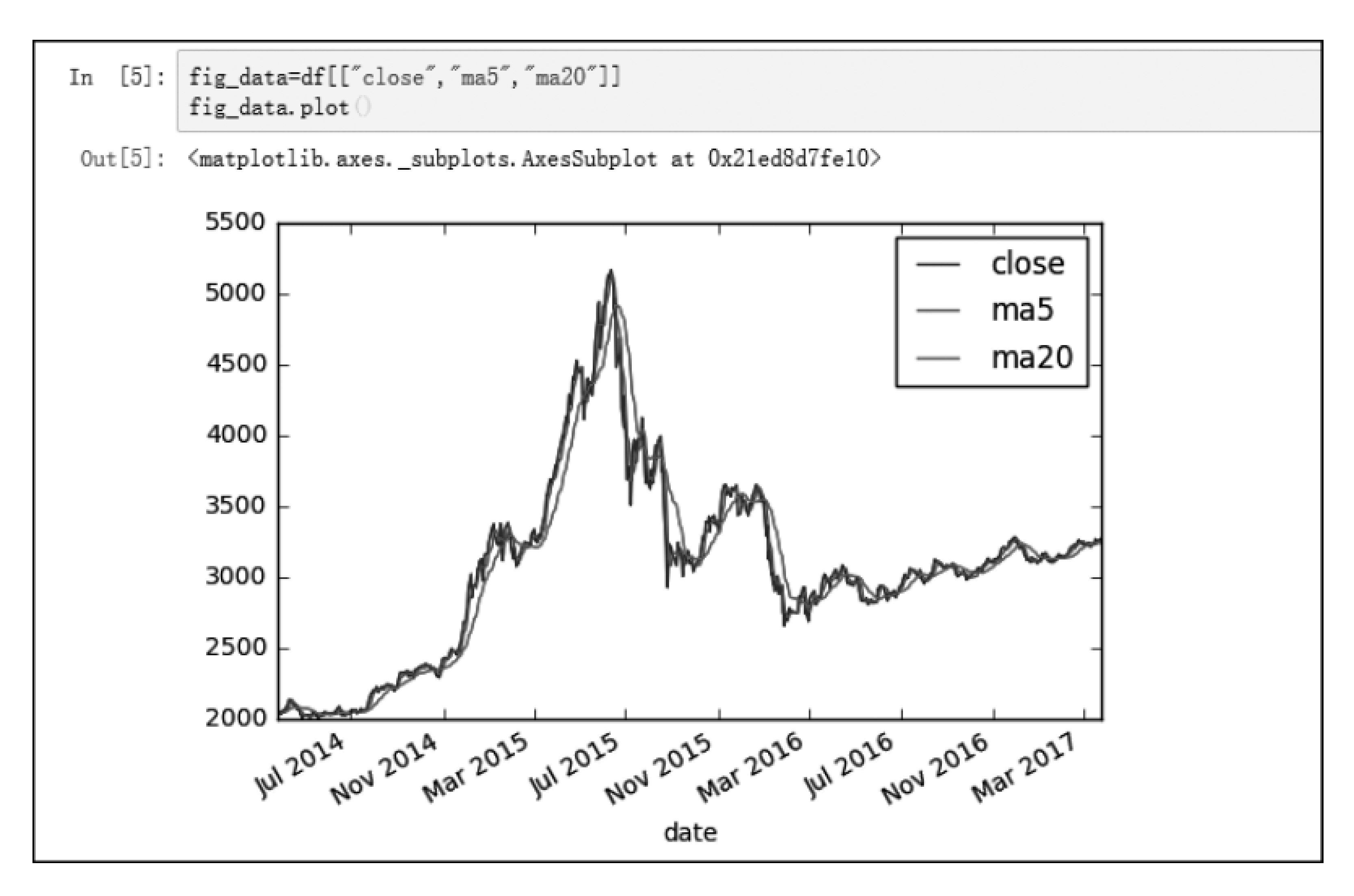


图 13-19

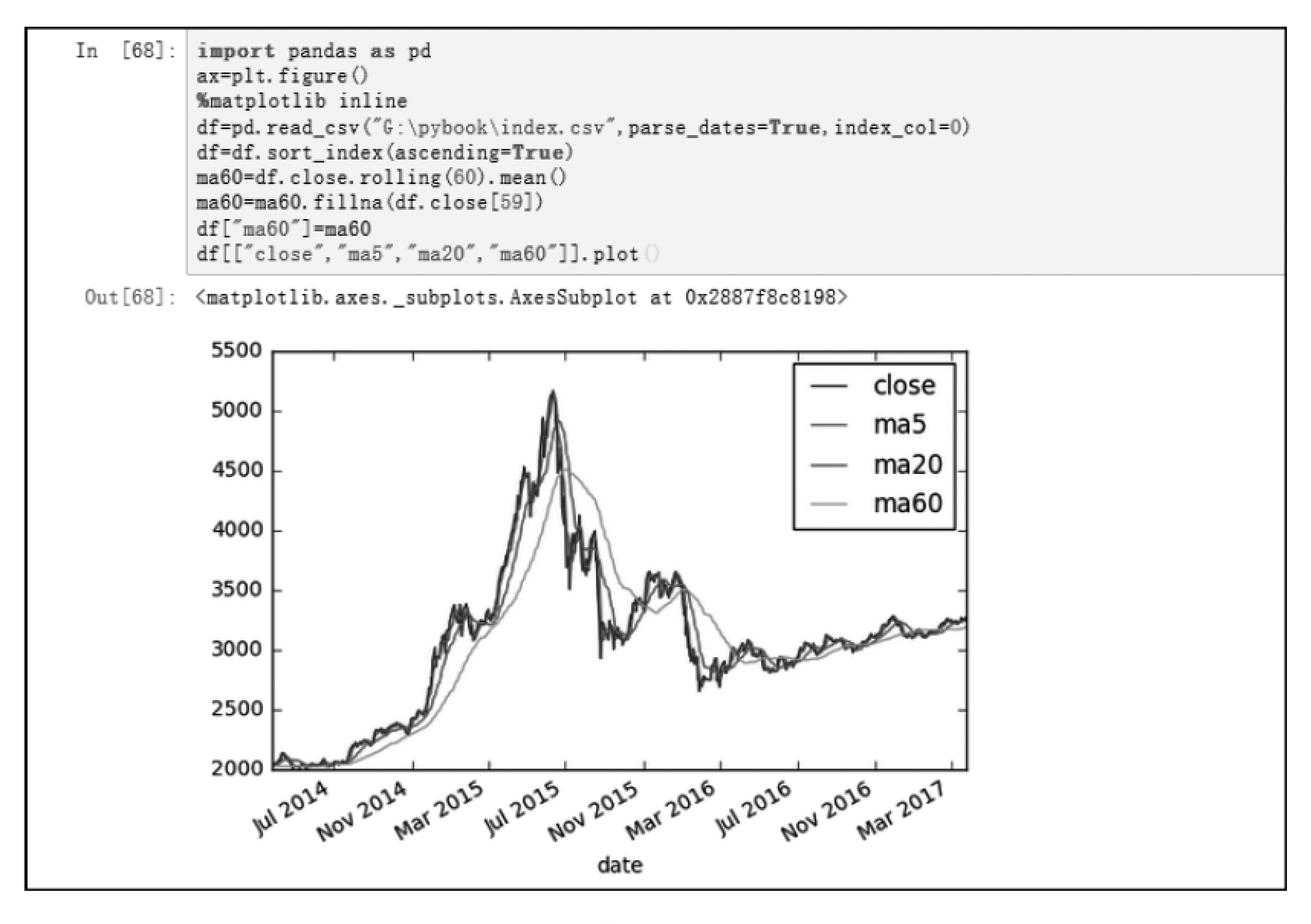


图 13-20

(3) df["ma60"]=ma60。

这个语句说明的是：把 ma60 数据添加到 df 对象上。

第 14 章 量化投资介绍

近年来，量化投资成了国内金融市场发展的热点，大家聚在一起不谈一下量化对冲就觉得好像不是做投资的一样。现在，量化投资和基本面分析、技术面分析可以说是三大主流方法。

1. 什么是量化投资

提起价值投资，我们都太熟悉了，大家都知道其代表人物是巴菲特。

提起量化投资，不得不提起华尔街传奇人物詹姆斯·西蒙斯(James Simons)。西蒙斯是世界级的数学家，也是资深的对冲基金经理之一。因其通过将数学理论巧妙融合到投资的实战之中，西蒙斯成了投资界中首屈一指的“模型先生”。由其运作的大奖章基金(Medallion)在1989—2009年的20年间，平均年收益率为35%，若算上44%的收益提成，则该基金实际的年化收益率可高达60%，比同期标普500指数年均回报率高出20多个百分点，即使相较金融大鳄索罗斯和股神巴菲特的操盘表现，也要遥遥领先十几个百分点。

西蒙斯通过将数学模型和投资策略相结合，逐步走上神坛，开创了由他扛旗的量化时代。西蒙斯做的事情就是量化投资。

量化投资就是利用计算机技术并且采用一定的数学模型去实践投资理念，实现投资策略的过程。

价值投资和技术分析是引领过去一个世纪的投资方法，随着计算机技术的发展，已有的投资方法和计算机技术相融合，产生了量化投资。

2. 量化投资 vs 传统投资

传统的股票投资主要有价值投资和技术分析，价值投资是看基本面的，而技术分析研究的是各种图形。与传统的投资方法不同，数量化投资不是以个人判断来管理资产，而是将投资经理的思想、经验和直觉反映在量化模型中，利用计算机帮助人脑处理大量信息，并进行投资决策。

对传统主动型投资人而言，决策广度的有限性，体现在跟踪股票数量上的限制，以及决策时思考变量上的限制；量化投资具有更大的投资视角和广度，能够快速高效地在全市场范围内进行海量信息处理和挖掘。当然，传统的主动投资方法在决策深度上是有优势的，所以，做更加深入的基本面研究，以弥补决策广度的不足是决定成败的关键。但是，随着市场信息传递速度的加快，众多分析师对基本面数据的不断挖掘、更加深入的分析似乎越来越难以弥补决策广度的不足。同时，传统投资的管理者本身情绪难免会受到周边环境的影响，常常会作出一些偏离自己判断的交易行为，而量化投资客观性、纪律性较强，按照严格的策略规则理性投资，能有效克服投资过程中的随意性和情绪化行为。

量化投资方法可以广泛应用在整个投资流程中，从自上而下的资产配置、行业配置、

风格配置，到自下而上的数量化选股等。

量化投资并不是传统投资方式的对立者，本质上只是一种投资手段，随着计算机技术的发展产生的一种投资手段。就像随着计算机技术的发展，大数据的概念也随之产生了。

量化基金的主要特点是将定性研究的理论通过数量模型演绎出来，借助计算机强大的处理信息的能力，全范围地筛选符合“标准”的股票，避免任何投资“盲点”的产生，最大限度地捕捉“标准”的投资对象。由于借助量化模型，定量投资能够避免基金经理情绪、偏好等对投资组合的干扰，精确地反映基金管理人的投资思想，最大限度地“理性”投资。

3. 量化投资的价值

量化投资对于基金公司/资产管理公司而言，有着非常明显的价值。

首先是容易冲规模。一个有效的量化模型是可以在多个产品上进行快速复制，从而迅速做大规模。这个在巴克莱的指数增强系列产品上得到了最明显的体现。截至 2011 年底，巴克莱量化基金，管理规模超过 1.6 万亿美金，超过富达基金，成为全球最大的资产管理公司。

其次是可以获得绝对收益。利用量化对冲方式，构建与市场涨跌无关的产品，赚取市场中性的策略，适合追求稳健收益的大机构客户，如保险资金、银行理财等。这个产品的代表性公司就是目前全球最大的对冲基金 Bridge Water，旗下的旗舰产品 Pure Alpha 过去 5 年共赚取超过 350 亿美金。

最后是杜绝了内幕消息和老鼠仓。量化投资只利用公开数据，通过数学模型的运算，挖掘出隐藏在公开数据后面的信息，从而战胜市场，从方法论上杜绝了内幕消息的可能。在交易过程中利用复杂的 IT 系统进行程序化交易，使得老鼠仓也无法成为可能。在国内金融市场监管日趋规范的情况下，量化投资这种方法必然会成为投资研究的主要方法。

4. 量化投资的理论基础

说到量化投资的理论基础，就要从市场有效性假说说起，技术分析、基本面分析和量化分析代表了有效市场的三个不同的层次。在无效市场中，技术分析是充分有效的，这在中国资本市场最初的 10 年得到了很好的体现；当市场进入弱有效市场后，可以依靠基本面分析获得超额收益，2000—2010 年这 10 年基本上属于这个时代；当市场进入半强有效市场后，也就是从 2010 年开始我们可以观察到大部分基本面分析的产品已经无法获得超额收益，此时国内市场已经进入半强有效市场。当然当市场进入强有效市场后，则无论哪种方法均无法战胜市场，那时候只能被动指数化投资。

传统的有效市场假说认为，在半强有效市场，只能依靠非公开信息（内幕消息或者私人消息）来获得超额收益。但是我们可以知道的是，除了非公开信息并不是只有内幕消息和私人消息外，还有一个获得非公开信息的方法：就是利用数据挖掘的方法，从公开的数据中挖掘出非公开信息，也就是量化投资的方法。这也就是在美国等成熟市场（基本上进入半强式有效市场状态），量化投资为什么可以得到蓬勃发展的原因。

随着资本市场有效性的提高，中国开始进入半强式有效市场阶段，再加上监管层对内幕消息的监管越来越严厉，使得通过这种方法获得非公开信息的方式越来越难，因此量化投资就成了一个最好的获得非公开信息的科学理论与技术。

通过上述对有效市场假说的分析，已经得到了明确的答案：量化投资是在半强式有效

市场中的最佳分析理论，也几乎是唯一可行的分析理论。

5. 量化投资的前景

中国经济经过 30 年的高速发展，各行各业基本上已经定型，能够让年轻人成长的空间越来越小了。未来 10 年，量化投资与对冲基金领域是少有的几个，可无论是出生贵贱，无论是学历高低，无论是有无经验，只要你勤奋、努力。脚踏实地地研究模型，研究市场，开发出适合市场稳健盈利的交易系统，实现财务自由，并非是遥不可及的梦想。

你只要好好研究量化模型，找到持续稳定盈利的策略，自然就会有大量的资金来找你合作，实现财务自由。到时候你就可以开着游艇出海，去拉斯维加斯享受，去非洲草原猎象，又何必在乎眼前的这点免费旅游呢。

在中国目前的很多领域，赚钱已经变成一个非常困难的事情，但是在量化投资与对冲基金领域，却完全可以依靠自己的勤奋与努力去赚钱。一个持续稳定赚取的模型，不是靠关系和背景就可以的，而是要靠着自己的聪明才智和脚踏实地的工作。

第 15 章

量化投资平台的介绍

15.1 量化投资的实务

量化投资是量化金融行业中最为尖端的一个领域，不论你是希望通过面试还是构建自己的交易策略，都会花费大量的时间与精力学习相关的知识。不仅如此，你还需要过硬的编程技术，至少需要精通一门高级编程语言（如 MATLAB、R 或 Python），而且伴随着高频交易策略的日益盛行，技术层面对于策略执行效果越来越至关重要，精通 C/C++ 也许是做高频交易最佳的选择。

量化投资系统包含以下四个部分：

（1）策略构建。策略构建即寻找策略，发掘可用优势，决定交易频率。

（2）策略回测。策略回测即获取数据，分析策略表现，排除模型偏差。

（3）交易执行系统。交易执行系统即对接经纪商，自动化交易，合理减少交易费用。

（4）风险管理。风险管理即最优化资产配置，根据凯利公式与交易心理学决定风险容忍度。

1. 策略构建

量化投资始于研究，研究过程包含选定策略、校验策略（检查其是否与当前投资组合的其他策略有冲突）、优化策略（通过数据对策略调优，提高回报率，降低风险）等过程。

获得策略的地方有很多，各个相关网站社区以及百度文库，如果使用 Google 会发现国外有很多不错的论文，比如 SSRN 网站的文章。

策略/策略集一旦构建完成，就需要用历史数据进行收益能力的评估测试。接下来就是回测发挥的环节了。

2. 策略回测

策略回测的目的在于使用历史及样本外数据对策略进行验证，确定其能否创造预期收益，结果也会被看作是该策略实盘操作的一个预期值。但实盘涉及因素极其复杂，回测并不能确保策略的成功。

回测可能是量化投资中要求最为精细的一环，因为太多可能的人为偏差会涉及其中，一般包括前视偏差、幸存偏差和优化偏差（也被称为数据透视偏差）。

通过历史数据测试及优化是回测必须的步骤。现在国内有好几家量化交易平台都支持回测，后面我们会介绍一些常见的平台。

进行回测时，回测系统一般会给出策略的收益表现。一般来讲，我们会关注几个常见的指标，比如 2～3 个，很多平台会提供几十个指标，具体情况读者可以自己权衡。下面我们介绍几个非常重要的指标。

（1）胜率。胜率反映的是盈利的概率，指的是在一个策略运行固定的周期内，在所有

的交易次数中，盈利的次数所占的百分比。

（2）总盈利/总亏损。总盈利/总亏损反映的是盈利和亏损的关系，如果盈利次数高，但每次盈利的金额很小，也会导致实际盈利不大，所以需要考察盈利和亏损的关系。

（3）夏普比率。夏普比率（sharpe ratio），又被称为夏普指数。夏普比率是一个可以同时对收益与风险加以综合考虑的三大经典指标之一。投资中有一个常规的特点，即投资标的的预期报酬越高，投资人所能忍受的波动风险越高；反之，预期报酬越低，波动风险也越低。这个指标重要性就在于刻画了收益和风险的关系。

（4）最大回撤率。在选定周期内任一历史时点往后推，产品净值走到最低点时的收益率回撤幅度的最大值。最大回撤用来描述买入产品后可能出现的最糟糕的情况。最大回撤是一个重要的风险指标，对于对冲基金和数量化策略交易，该指标比波动率还重要。

大家可以看到前两个是反应盈利的指标，后两个是反应风险的指标。事实上，读者随着不断的熟练或者交易策略本身的要求会选择不同的指标体系进行评价。

如果对于回测，针对回测的评价体系也符合预期，那么就可以构建交易执行系统了。

3. 交易执行系统

通过交易执行系统，策略产生的交易指令被传送给经纪商并执行。现实当中，指令生成可以是半自动或者全自动的，执行机制也可以配置为手工、半手工或者完全自动。低频交易中，手工和半手工较为常见，而高频交易则需要建立完全自动化的执行机制，通常也需要与指令生成模块紧密配合。

构建交易执行系统的关键在于对接券商的接口、降低交易费用的策略（包含佣金、滑点及经纪商买卖差价）及处理实盘与回测预测业绩差异的处理。

执行过程中经常会出现问题的地方在于交易费用的优化，最为常见的费用包括以下三类：

（1）佣金/税费（commission/tax）。佣金/税费通常由经纪商、交易所收取。

（2）滑点（slippage）。滑点指订单希望的成交价格与实际成交价格的差异（一般会在行情波动大或市场缺乏流动性的时候出现）。

（3）价差（spread）。价差指交易标的的买卖报价之差，注意价差并非常量，它会跟随市场订单的情况而不停变化。这个在高频交易中非常明显，对收益的影响也非常显著。

交易成本对策略影响很大，甚至可以让一个原本高盈利、高夏普率的策略摇身一变成为很糟糕的策略，通过回测正确预测交易成本极具挑战性。依据交易策略频率，需要获取历史成交数据（包含逐笔交易的买卖信息），数据量很大，所以大型基金中，基本所有宽客团队都在专注于指令的执行优化。

交易执行系统最后一个主要关注点在于策略的真实表现与回测中的期望的差别，有许多原因都会导致这种情况，比如之前在回测部分已经提到的前视偏差和数据透视偏差。但一些策略在真实运行前很难发现偏差，高频交易尤为明显。交易策略和指令执行系统都可能存在回测中无法复现，但在实盘中会出现一些预想不到的错误，市场也可能因为一个策略的部署从而天翻地覆。

新的监管环境，投资者情绪与宏观经济的变化都可能导致市场行为的变化，进而也会影响策略的表现。所以策略的不断更新和优化是一个烦琐和累积的过程。

4. 风险管理

做交易，新手和老手最大的区别可能就是对风险管理的态度了，因为这个问题很容易理解，但是很难控制，特别是作为一个交易身临其境，面对金钱那么近，面对行情波动的时候，那种心情就像足球场上看球的和踢球的一样。

常见的风险有技术性风险（比如交易所的服务系统突然发生了硬盘故障）、经纪商风险（比如经纪商网络出现问题）、托管的服务器网络故障等。简言之，只要影响交易进行的因素，就可能带来风险。

15.2 量化投资平台

我们这里介绍的量化投资平台主要指的是可以通过 Python 进行编程实现的量化投资平台，所以不会包括金字塔、TB 或者 MC 这些知名的量化投资平台。

1. 数据支持

数据方面，我们推荐 Wind 以及 TuShare。

（1） Wind

Wind 在国内的地位就相当于国外的彭博，数据全是第一个优势。

在国内市场，Wind 资讯的客户包括中国绝大多数的证券公司、基金管理公司、保险公司、银行和投资公司等金融企业；在国际市场，已经被中国证监会批准的合格境外机构投资者（QFII）中的众多机构是 Wind 资讯的客户。

在金融财经数据领域，Wind 资讯已建成国内完整的、准确的以金融证券数据为核心一流的大型金融工程和财经数据仓库，数据内容涵盖股票、基金、债券、外汇、保险、期货、金融衍生品、现货交易、宏观经济、财经新闻等领域，新的信息内容及时进行更新以满足机构投资者的需求。

第二个优势也是我们要重点介绍的，Wind 有一个大奖章的网站（http://snap.windin.com/dajiangzhang/），主要是针对 Wind 提供的各种获取数据的接口，如图 15-1



图 15-1

读者可以通过 Python 的接口获取 Wind 的数据并进行更精细、个性化的数据分析，

并保存数据、建模等。

（2）TuShare

关于 TuShare 我们就不再介绍了，相对来说，TuShare 更好使用，考虑到免费以及对数据需求不是那么深入的话，推荐就直接使用 TuShare。

2. 量化投资平台

量化平台可以看作是一个已经搭建好的框架。用户只需添加一些自己的买卖条件，即可回测策略，免去了自己从无到有搭建基础框架的过程。也就是说，通过量化投资平台，我们只是需要做的是构建策略和回测。

下面我们主要介绍优矿、聚宽、米宽和京东量化金融平台。它们都借鉴于 quantopian，所以整体框架都差不多，下面就以特色做介绍。

（1）优矿（网址：https://uqer.io/home/）

① 依靠通联集团，引入了海量数据，数据方面有优势。

② 最大的亮点是引入了 jpyter notebook，同时需要使用 Google 浏览器。

（2）聚宽（网址：https://www.joinquant.com/）

① 策略信号微信通知。

② 适合一般的量化投资者爱好者，容易上手。

（3）米宽（网址：https://www.ricequant.com/? f=n）

① 对数据的使用比较方便。

② 适合有编程基础的读者。

（4）京东量化金融（网址：https://quant.jd.com/）

① 依靠京东集团资源，情景化强，对于在京东购物和理财的用户，使用京东量化金融平台是不错的选择。

② 支持 Java，对于那些愿意用 Java 的是一个利好。

以上这些观点，主观性很强，个人建议还是看个人喜好，然后关于策略和一些知识的学习各个平台都有一些不错的资源，建议都去逛逛。熟悉一个就好，如果工作需要，转用其他的，都是差不多的。

第 16 章 量化策略

16.1 量化策略概述

1. 什么是策略?

策略,可以实现目标的方案集合;在证券交易中,策略是指当预先设定的事件或信号发生时,就采取相应的交易动作。

2. 什么是量化策略?

量化策略是指使用计算机作为工具,通过一套固定的逻辑来分析、判断和决策。量化策略既可以自动执行,也可以人工执行。

3. 一个完整的量化策略构建

一个完整的策略需要包含输入、策略处理逻辑、输出;策略处理逻辑需要考虑选股、择时、仓位管理和止盈止损等因素。

16.2 常见的量化交易策略

量化策略很多,我们会发现高校学生的优势可能就在于对算法的研究,因为在学校不像在市场上参与实务操作,有市场经验和感觉,但是对于一些理论研究反而没有边界,设计的一些算法可能还是不错的策略。我们这里介绍的策略也是基于一种分类,比如对于一个交易标的,可分为均值回归和动量策略;对于有两只交易标的的,可用配对交易或者统计套利;对于有多个交易标的的,可有多因子选股模型。接下来,我们将分别予以介绍。

1. 双均线模型

移动平均(moving average,MA),又称“移动平均线”,简称均线,是技术分析中一种分析时间序列数据的工具。最常见的是利用股价、回报或交易量等变量计算出移动平均。均线是最基础的动量指标,一般短期均线上穿长期均线意味着近期买盘较强势,可以作为买入信号,俗称“金叉”;反之短期均线由上向下穿破长期均线意味着近期卖盘较强势,可以作为卖出信号,俗称“死叉”。

2. 均值回归

先讲一个故事。从前有座山,山里有座庙,庙里有一个老和尚,这个老和尚是个佛法高深的住持。山下不远处有一家证券公司,跟寺庙遥遥相对。一天,庙里来了许多炒股的人,在菩萨面前烧了许多香,求菩萨保佑他们脱离苦海。

老和尚心善,问是怎么回事。香客们说:“股票大跌,我们被深度套牢,赔了很多钱,不知怎么才能脱身。”老和尚心想:“我不入地狱,谁入地狱?”于是倾其所有,买进股票。

过了一段时间,股市大涨,一股难求,香客们又来庙中烧香,为买不到股票发愁。老和

尚念道："善哉，善哉，先天下之忧而忧，后天下之乐而乐。"于是把所有股票都卖个精光。

几个来回，老和尚赚了很多钱，大家纷纷向老和尚讨教炒股秘诀。老和尚说："哪有什么秘诀，我只是无欲无求，抱着一颗善心而已！"

老和尚用的就是均值回归策略。

我们可以把均值回归看作是资本市场的万有引力定律，就像世事万物都有"分久必合，合久必分"的一样。

3. 配对交易

无论市场是牛市、熊市，配对交易都可以在高频交易中帮投资者获得一笔不菲且稳定的收益。这类配对方法主要是通过做多一只股票，同时做空另一只与其存在某种相关性的股票而获得平稳收益。

量化策略实现步骤如下：

(1) 找到相关性高的一对股票

直观上来看，我们需要寻找的是两只在公司主营业务、规模大小甚至是风险因素等方面要有一定的相似性的股票，可以在同一行业或者相关联行业（如房地产和钢铁行业）中寻找。另外，还可以通过编写程序自动查找近 N 日内相关系数最大的或存在协整关系的一对股票（可采用 Pearson 相关系数计算、Johansen's 协整检验等）。例如：计算一段时间某行业内所有股票价格的相关性矩阵，选取相关系数最大的两只股票。

(2) 计算时间段内股价间稳定关系

设主流方法是寻找到对数股价间的协整关系，协整关系的检验又包括两个子过程，首先是确定线性关系；其次是残差（residual）序列的平稳性检验。

(3) 交易规则的制定

交易规则的制定主要包括触发机制的确定和止损机制的确定。价差序列是按统计规律依概率收敛而非必然收敛，所以在交易过程中可能会遇到价差序列发散的情况，如配对组合中股票的基本面突然发生变化等，在带来巨额收益率的同时也会出现巨额的亏损，因此对于配对交易策略，风险控制显得尤为重要。

4. 多因子选股

多因子模型是应用最广泛的一种选股模型，基本原理是采用一系列的因子作为选股标准，满足这些因子的股票则被买入，不满足的则被卖出。

举一个简单的例子：如果有一批人参加马拉松比赛，想要知道哪些人会跑到平均成绩之上，那只需在跑前做一个身体测试即可。那些健康指标靠前的运动员，获得超越平均成绩的可能性较大。多因子模型的原理与此类似，我们只要找到那些对企业的收益率最相关的因子即可。

各种多因子模型核心的区别第一是在因子的选取上，第二是在如何用多因子综合得到一个最终的判断。

一般而言，多因子选股模型有两种判断方法，一是打分法，二是回归法。

打分法就是根据各个因子的大小对股票进行打分，然后按照一定的权重加权得到一

个总分，根据总分再对股票进行筛选。回归法就是用过去的股票的收益率对多因子进行回归，得到一个回归方程，然后再把最新的因子值代入回归方程得到一个对未来股票收益的预判，然后再以此为依据进行选股。

多因子选股模型的建立过程主要分为候选因子的选取、选股因子有效性的检验、有效但冗余因子的剔除、综合评分模型的建立和模型的评价及持续改进五个步骤。

第 17 章 开启你的量化投资之旅

说起量化交易入门，很多时候得到的答案都是长长的书单，让人很茫然，即使你在网上买了很多书，但要认认真真读完一本的都可能很少。

我们这里就通过使用聚宽进行快速入门。整个流程包括数据获取、策略回测、行情链接和交易信号。

我们通过学会写一个简单的量化交易策略，建立一个模型，充分理解策略的基本框架，学会建立连接实盘的模拟交易，并使其自动发送交易的信号到微信，通过微信建立跟进自己的策略效果。

首先，进入 JoinQuant，单击导航栏我的策略，新建策略，进入策略编辑画面，如下图 17-1 所示。



图 17-1

图中左侧是编写策略代码，右侧是策略运行结果。我们就在左侧写策略代码。

下面我们就写个量化交易策略——单股票均线策略。

1. 构建策略

若昨日收盘价高出过去 5 日平均价，今天开盘买入股票；若昨日收盘价低于过去 5 日平均价，则今天开盘卖出股票。

每天看看昨日收盘价是否高出过去 5 日平均价，是的话开盘就买入，不是的话开盘就卖出。每天都这么做，循环下去。

对应代码也是这两个部分：

```
def initialize(context):
```

用来写最开始要做什么的地方。

```
def handle_data(context,data):
```

用来写每天循环要做什么的地方。

几乎所有策略都基于这个基本的策略框架：先初始化，然后循环操作。

（1）初始化。初始化即最开始要做的事情，如设定沪深300作为基准、开启动态复权模式（真实价格）、输出内容到日志上、交易时的手续费等。

（2）周期循环。周期循环即每个周期要做的事情，如计算指标，买入卖出等，周期可能是分钟、天等，本文策略的周期是一天。当你要做一些盘中短线操作的时候，周期就要调成分钟（先别着急，会遇到的）。

我们写设置时要交易的股票代码，比如代码中的平安银行（000001）

```
def initialize(context):
```

g.security='000001.XSHE'# 存入平安银行的股票代码。

"g."是什么？全局变量前都要写"g."，全局变量就是全局都能用的变量，一般变量只能在该函数下使用。例如，security不加"g."，只能在第一部分即initialize里用。

"XSHE"是什么？股票代码使用时要加后缀，深交所股票代码后缀为".XSHE"，上交所股票代码后缀为".XSHG"。

代码中"#"是什么？"#"后的内容都是注释，是为代码做说明的，不会被计算机当作代码处理。这个在前面Python中已有说明。

2. 获取收盘价与均价

首先，获取昨日股票的收盘价。

用法：变量 = data[股票代码].close。

```
#取得昨日股票价格
    current_price=close_data['close'][-1]
    然后，#获取股票的收盘价
    close_data=attribute_history(security, 5, '1d', ['close'])
    #取得过去五天的平均价格
    MA5=close_data['close'].mean()
    #取得上一时间点价格
    current_price=close_data['close'][-1]
```

3. 判断是否买卖

数据都获取完，该做买卖判断了。这里用多少钱买入，我们需要设置资金。

```
#如果上一时间点价格高出五天平均价 1%，则全仓买入
    if current_price>1.01*MA5
#如果上一时间点价格低于五天平均价，则空仓卖出
    ifcurrent_price<MA5
```

4. 买入卖出

```
#取得当前的现金
    cash=context.portfolio.available_cash
```

```
#用法：order_value(要买入股票股票的股票代码,要多少钱去买)
order_value(g.security, cash)#用当前所有资金买入股票
#用法：order_target(要买卖股票的股票代码,目标持仓金额)
order_target(g.security, 0)#将股票仓位调整到 0,即全卖出。
```

(1) 为什么没有指定交易价格？此策略是按天回测进行的，且使用较为简单的市价单下单方法，交易价格为开盘价(加上滑点)。

(2) 无法交易的情况？涨跌停，停牌，(T+1)制度等无法交易的情况，系统会自动使下单不成交并在日志中发出警告。

5. 策略代码写完，进行回测

具体的代码如图 17-2 所示，左侧已经是一个很具体的模板了，读者可以自行阅读，熟悉。

现在，在策略回测界面右上部，设置回测时间从 2016-09-01 到 2017-03-01，设置初始资金 100 000，设置回测频率，然后单击“运行回测”按钮。



图　17-2

6. 建立模拟交易，使策略和行情实时连接，自动运行

策略写好，回测完成，单击回测结果界面，如图 17-3 所示。



图　17-3

7. 开启微信通知，接收交易信号

单击聚宽导航栏我的交易，可以看到创建的模拟交易。

单击右边的微信通知开关，将 OFF 调到 ON，按照指示扫描二维码，绑定微信，就能用微信接收交易信号了，如图 17-4 所示。

模拟交易列表 + 新建模拟交易 删除 迎新年，模拟交易使用权5折大优惠

全部 进行中 已关闭

名称	频率	状态	分享	开始时间	停止时间	收益	今日收益	最大回撤	微信通知	时限
模拟交易	每天	未开始		2017-04-08		--	--	--	ON	永久
高息股策略	每天	进行中		2016-12-19		7.7%	-0.08%	1.74%	ON	永久

图 17-4

相信读者会发现大部分的量化交易策略都是这样的流程，先建立一个模型，然后不断熟悉，遇到问题不断查询和交流，然后应用其他的策略进行验证和学习。

7. 开启微信通知，接收交易信号

单击聚宽导航栏我的交易，可以看到创建的模拟交易。

单击右边的微信通知开关，将 OFF 调到 ON，按照指示扫描二维码，绑定微信，就能用微信接收交易信号了，如图 17-4 所示。

模拟交易列表　+ 新建模拟交易　删除　　迎新年，模拟交易使用权5折大优

全部　进行中　已关闭

名称	频率	状态	分享	开始时间	停止时间	收益	今日收益	最大回撤	微信通知	时限
模拟交易	每天	未开始		2017-04-08		--	--	--	ON	永久
高息股策略	每天	进行中		2016-12-19		7.7%	-0.08%	1.74%	ON	永久

图　17-4

相信读者会发现大部分的量化交易策略都是这样的流程，先建立一个模型，然后不断熟悉，遇到问题不断查询和交流，然后应用其他的策略进行验证和学习。

后　　记

读书和赚钱，读书也许仅仅是一种任务，一种兴趣，一种信仰，你可以说是为了文化传承；赚钱可是很客观的，读书和赚钱的相关性并没有我们想象的那么强。

科研和考试，科研也许是一种兴趣，一种信仰，考试就是为了能否通过，考试更讲究题海战术，讲究技术。

我写这本书也不是希望你要去研究 Python 这门语言，我们的目的很简单，就是希望你在量化投资或者量化金融领域能知道这门语言，掌握这门语言解决一些数据分析的常见问题，就像你使用 Excel 一样。也许，可以这么说，很久以前，办公需要掌握的是 Excel；但是现在你需要掌握的是 Python。

这本书既没有长篇大论，也没有介绍的那么详细，不是针对所有人的。我们需要的是一幅地图，掌握这门语言的一幅地图，毕竟我们的百宝箱——百度是很强大的。我们需要的是一个模型，毕竟问题总是很具体的、多样的、复杂的；而我们掌握的模型能够解决遇到的大部分问题。